In The United States Patent and Trademark Office
On Appeal From The Examiner To The Board
of Patent Appeals and Interferences

In re Application of:	Shriniwas Lohia
Serial No.:	09/436,920
Filing Date:	November 9, 1999
Confirmation No.	7304
Group Art Unit:	2141
Examiner:	Adnan M. Mirza
Title:	SYSTEM FOR COMMUNICATION MANAGEMENT INFORMATION AND METHOD OF OPERATION

MAIL STOP: APPEAL BRIEF - PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

Dear Sir:

Appeal Brief

Appellant has appealed to the Board of Patent Appeals and Interferences (the "Board") from the decision of the Examiner mailed January 18, 2007 (the "*Final Office Action*"), finally rejecting all pending Claims 1-26. Appellant filed a Notice of Appeal, with the statutory fee of $500.00, on April 18, 2007. Appellant respectfully submits this Appeal Brief with the statutory fee of $500.00.

Real Party in Interest

Appellant believes this Application is subject to assignment from the inventor to Cisco Technology, Inc.

Related Appeals and Interferences

Appellant previously appealed examination of the present Application to the Board through the filing of an appeal brief on January 22, 2004. That appeal was assigned Appeal No. 20005-0521. The Board affirmed in-part and overturned in-part the rejections then pending in a decision issued on May 24, 2005. A Request for Rehearing was denied on September 15, 2005. Copies of the Board's decision on the appeal and the rehearing are attached as Appendix D.

Status of Claims

Claims 1-26 are pending in this Application. Claims 1-26 stand rejected pursuant to the *Final Office Action*, and are all presented for appeal. All pending claims are shown in Appendix A, along with an indication of the status of those claims.

Status of Amendments

All amendments submitted by Appellant have been entered by the Examiner prior to the mailing of *Final Office Action*.

Summary of Claimed Subject Matter

This invention relates in general to data communication, and more particularly to a system for communicating management information (Page 1, Lines 2-4).[1]

Network devices are generally arranged in a chassis or housing at a particular location in a communication system. Each network device arranged in a particular chassis may be configured, initialized, or otherwise managed using consoles external to the chassis. A drawback to prior systems is that each network device in a chassis requires a corresponding, dedicated console to handle the management operations of the associated network device. A further drawback is that the interface card of each network device in the chassis must maintain a dedicated connection to its corresponding console. Such a configuration of consoles and network devices adds costs and complexities to the management operations of communication systems. (Page 2, Lines 2-14).

Thus, the present Application discloses a management system 10 that includes a management card 12 coupled to a number of interface cards 14 using links 16 and coupled to a client 18 using a link 20. In general, client 18 handles the primary management responsibilities for each of interface cards 14 and/or associated network devices 24 using management card 12. (Page 6, Lines 2-8).

Management card comprises any suitable combination of hardware and software components that establish one or more communication links 22 between client 18 and one or more interface cards 14 selected in response to a command 40, and communicate management information 42 to the interface cards 14 using communication links 22. Although management card 12 is illustrated separate from interface cards 14, it should be understood that management card 12 may be formed integral to or separate from a particular interface card 14. Management card 12 is described in greater detail with respect to FIGURE 2. (Page 6, Lines 9-19).

Management card 12 may establish and maintain communication links 22 between a client 18 and any number and combination of interface cards 14. A communication link 22 comprises any switched communication path that couples client 18 to an interface card 14 and communicates management information 42 using any suitable communication protocols, standards, and/or formats. (Page 6, Lines 23-30).

Each interface card 14 comprises any suitable combination of hardware and software components that enable network devices 24 to communicate with various components of a

[1] All citations in this section of the Appeal Brief are to Appellant's originally-filed Specification.

communication network (not explicitly shown) and with other components of system 10. For example, interface cards 14 include management ports 26 that couple network devices 24 to management card 12 using links 16 such that devices 24 may communicate with management card 12. Network devices 24 comprise computers, servers, workstations, IP telephones, routers, bridges, switches, gateways, hubs, and any other suitable electronic devices that may be managed by client 18 using management card 12. (Page 7, Lines 14-25).

In operation, a user operates client 18 to communicate a command 40 and management information 42 to management card 12. Management card 12 receives command 40 communicated by client 18 and establishes one or more communication links 22 between client 18 and particular interface cards 14 selected in response to command 40. Using communication links 22, management card 12 communicates to the particular interface cards 14, the management information 42 communicated by client 18. (Page 8, Lines 18-26).

FIGURE 2 illustrates management card 12 in more detail. Communication link 20 couples client 18 to interface 50. Interface 50 couples to a line driver 52, which in turn couples to a processor 54. Processor 54 manages the overall operation of management card 12 and couples to memory 56 and line driver 58. Line driver 58 couples to switch 60 at a first port 62. Links 16 couple to switch 50 at second ports 64. (Page 9, Lines 3-10).

Processor 54 comprises a central processing unit having any suitable general purpose data processing capabilities. Memory 56 comprises any suitable volatile or non-volatile memory device associated with processor 54. Memory 56 generally stores a number of files, lists, tables, or any other arrangement of information that support establishing communication links 22 and communicating management information 42 using communication links 22. For example, memory 56 stores program instructions 70 and a mapping table 72. Information maintained in memory 56 may reside in different components of management card 12 or in components external to management card 12. (Page 9, Line 28 - Page 10, Line 4).

Switch 60 comprises any suitable combination of hardware and software components that establish communication links 22 between clients 18 and particular interface cards 14 to direct, couple, or otherwise switch management information 42 communicated by client 18 to the appropriate interface cards 14. Switch 60 generally comprises a first port 62 coupled to client 18 and second ports 64 coupled to interface cards 14 using links 16. As described with respect to FIGURE 3, memory 56 stores a mapping table 72 associating particular interface

cards 14 and/or network devices 24 to particular ports 64 to support establishing communication links 22. (Page 10, Lines 22-34).

FIGURE 3 illustrates the contents of mapping table 72 stored in memory 56 of management card 12. Each entry of mapping table 72 includes mapping information, such as a device identifier 80, an operating system identifier 82, and a port identifier 84. Device identifier 80 comprises information identifying a particular network device 24 and/or an associated interface card 14. Operating system identifier 82 comprises information identifying the operating system, if any, associated with a corresponding network device 24. Port identifier 84 comprises information identifying a port 64 of switch 60 associated with a corresponding network device 24 and/or interface card 14. (Page 10, Line 34 - Page 11, Line 10).

In operation, processor 54 of management card 12 executes program 70 to support the communication of management information 42 to one or more interface cards 14. In one embodiment, processor 54 executes program 70 to support generating a command 40 and management information 42 at client 18. Interface 50 of management card 12 receives a command 40 communicated by client 18. Command 40 generally identifies one or more interface cards 14 to which management information 42 is directed and is generally communicated using a communication format associated with link 20. Line driver 52 converts command 40 from the communication format associated with link 20 to a communication format associated with processor 54. (Page 11, Lines 11-23).

Processor 54 receives command 40 identifying one or more particular interface cards 14 and determines the appropriate ports 64 of switch 60 associated with the particular interface cards 14 using the information stored in mapping table 72. In a particular embodiment, processor 54 also uses the information stored in mapping table 72 to configure management information 42 according to the operating systems of the network devices 24 associated with the particular interface cards 14 identified in command 40. (Page 11, Lines 24-32).

Processor 54 commands switch 60 to establish communication links 22 between client 18 and the determined ports 64 using line driver 58. Line driver 58 converts management information 42 to a communication format associated with links 16, interface cards 14, and/or network devices 24. Switch 60 establishes one or more communication links 22 between the appropriate ports 62 and 64 to couple client 18 to the appropriate interface cards 14.

Management card 12 communicates management information 42 using communication link 22. In particular, management card 12 may communicate management information 42 from client 18 to interface cards 14 and/or may communicate management information 42 from interface cards 14 to client 18 such as, for example, in response to a request. (Page 11, Line 33 - Page 12, Line 10).

With regard to the independent claims currently under Appeal, Appellant provides the following concise explanation of the subject matter recited in the claim elements. For brevity, Appellant does not necessarily identify every portion of the Specification and drawings relevant to the recited claim elements. Additionally, this explanation should not be used to limit Appellant's claims but is intended to assist the Board in considering the Appeal of this Application.

For example, independent Claim 1 recites the following:

> A system for communicating management information, comprising:
> a first interface card (e.g., Figure 1; Page 8, Lines 15-27; Page 9, Lines 28-30; Page 9, Lines 16-22; Page 14, Lines 28-34);
> a second interface card (e.g., Figure 1; Page 8, Lines 15-27; Page 9, Lines 28-30; Page 9, Lines 16-22; Page 14, Lines 28-34); and
> a management card coupled to the first interface card and the second interface card (e.g., Figure 1; Page 7, Lines 8-19; Figure 2; Page 10, Line 9 - Page 12, Line 6; Figure 3; Page 12, Line 7 - Page 14; Line 2) the management card operable to:
>
> > receive a command from a client, the command identifying an interface card or a network device associated with an interface card; (e.g., Figure 1; Page 7, Line 31 - Page 8, Line 3; Page 9, Lines 22-27; Page 10, Lines 30-34; Page 12, Lines 20 - Page 13, Line 7; Figure 4; Page 14, Lines 3-15);
> >
> > establish a communication link between the client and a particular one of the first interface card and the second interface card selected in response to the command communicated by the client, wherein the communication link forms a complete path that couples at least the client to at least the particular interface card (e.g., Figure 1; Page 7, Lines 20-30; Page 7, Line 28 - Page 9, Line 3; Page 9, Lines 24-27; Figure 2; Page 10, Lines 10-11; Page 11, Line 30 - Page 12, Line 6; Page 12, Line 34 - Page 13, Line 16; Figure 4, Lines 16-20); and
> >
> > communicate management information using the communication link (e.g., Figure 1; Page 8, Lines 4-14; Page 9, Line 28 - Page 10, Line 8; Page 11, Lines 3-7; Page 11; Line 30 - Page 12; Line 2; Page 13, Lines 8-22; Figure 4; Page 14, Lines 20-34).

As another example, independent Claim 7 recites the following:

A method for communicating management information performed by a management card, comprising:

receiving a command from a client, the command identifying a particular one of a first interface card and a second interface card (e.g., Figure 1; Page 7, Line 31 - Page 8, Line 3; Page 9, Lines 22-27; Page 10, Lines 30-34; Page 12, Lines 20 - Page 13, Line 7; Figure 4; Page 14, Lines 3-15);

establishing a communication link between the client and the particular interface card in response to receiving the command, wherein the communication link forms a complete path that couples at least the client to at least the particular interface card (e.g., Figure 1; Page 7, Lines 20-30; Page 7, Line 28 - Page 9, Line 3; Page 9, Lines 24-27; Figure 2; Page 10, Lines 10-11; Page 11, Line 30 - Page 12, Line 6; Page 12, Line 34 - Page 13, Line 16; Figure 4, Lines 16-20); and

communicating management information using the communication link (e.g., Figure 1; Page 8, Lines 4-14; Page 9, Line 28 - Page 10, Line 8; Page 11, Lines 3-7; Page 11; Line 30 - Page 12; Line 2; Page 13, Lines 8-22; Figure 4; Page 14, Lines 20-34).

The citations listed above with respect to independent Claim 7 are also applicable to independent Claim 21, which is directed to a system.

As yet another example, independent Claim 14 recites:

A management card, comprising:

a switch coupled to a first interface card and a second interface card (Figure 2; Page 10, Lines 14-16; Page 11, Line 30 - Page 12, Line 6); and

a processor coupled to the switch (Figure 2; Page 10, Lines 9-16; Page 10, Line 35 - Page 1, Line 1) and operable to:

receive a command communicated by a client, the command identifying a particular one of the first interface card and the second interface card (e.g., Figure 1; Page 7, Line 31 - Page 8, Line 3; Page 9, Lines 22-27; Page 10, Lines 30-34; Page 12, Lines 20 - Page 13, Line 7; Figure 4; Page 14, Lines 3-15); and

command the switch to establish a communication link between the client and the particular interface card, wherein the communication link forms a complete path that couples at least the client to at least the particular interface card (e.g., Figure 1; Page 7, Lines 20-30; Page 7, Line 28 - Page 9, Line 3; Page 9, Lines 24-27; Figure 2; Page 10, Lines 10-11; Page 11, Line 30 - Page 12, Line 6; Page 12, Line 34 - Page 13, Line 16; Figure 4, Lines 16-20).

As yet another example, independent Claim 22 recites:

A system for communicating management information, comprising:

a first interface card coupled to a first network device (e.g., Figure 1; Page 8, Lines 15-27; Page 9, Lines 28-30; Page 9, Lines 16-22; Page 14, Lines 28-34) that uses a first operating system (Figure 3; Page 12, Lines 7-15; Page 12, Line 34 - Page 13, Line 7; Page 13, Line 23 - Page 14, Line 2; Page 14, Lines 16-34);

a second interface card coupled to a second network device (e.g., Figure 1; Page 8, Lines 15-27; Page 9, Lines 28-30; Page 9, Lines 16-22; Page 14, Lines 28-34) that uses a second operating system (Figure 3; Page 12, Lines 7-15; Page 12, Line 34 - Page 13, Line 7; Page 13, Line 23 - Page 14, Line 2; Page 14, Lines 16-34); and

a management card coupled to the first interface card and the second interface card (e.g., Figure 1; Page 7, Lines 8-19; Figure 2; Page 10, Line 9 - Page 12, Line 6; Figure 3; Page 12, Line 7 - Page 14; Line 2), the management card operable to:

receive a command from a client, the command identifying an interface card or a network device associated with an interface card (e.g., Figure 1; Page 7, Line 31 - Page 8, Line 3; Page 9, Lines 22-27; Page 10, Lines 30-34; Page 12, Lines 20 - Page 13, Line 7; Figure 4; Page 14, Lines 3-15);

establish a communication link between the client and a particular one of the first interface card and the second interface card selected in response to the command communicated by the client (e.g., Figure 1; Page 7, Lines 20-30; Page 7, Line 28 - Page 9, Line 3; Page 9, Lines 24-27; Figure 2; Page 10, Lines 10-11; Page 11, Line 30 - Page 12, Line 6; Page 12, Line 34 - Page 13, Line 16; Figure 4, Lines 16-20);

configure management information for the operating system of the network device associated with the particular interface card (e.g., Figure 1; Page 13, Lines 2-7; Figure 4; Page 14; and

communicate the management information using the communication link (e.g., Figure 1; Page 8, Lines 4-14; Page 9, Line 28 - Page 10, Line 8; Page 11, Lines 3-7; Page 11; Line 30 - Page 12; Line 2; Page 13, Lines 8-22; Figure 4; Page 14, Lines 20-34).

Grounds of Rejection to be Reviewed on Appeal

1. Do Claims 1, 7, 14, and 21 comply with 35 U.S.C. § 112, first paragraph?

2. Are Claims 1, 4-7, 10-14, 16, 18-22, and 24-26 patentable under 35 U.S.C. § 102(e) over U.S. Patent No. 4,937,777 issued to Flood et al ("*Flood*")?

3. Are Claims 2-3, 8-9, 15, 17, and 23 patentable under 35 U.S.C. § 103(a) over *Flood* in view of U.S. Patent No. 6,304,895 issued to Schneider et al. ("*Schneider*")?

Argument

The rejection of Claims 1, 7, 14, and 21 under 35 U.S.C. § 112, first paragraph, is improper and should be reversed by the Board. The rejection of Claims 1, 4-7, 10-14, 16, 18-22, and 24-26 as patentable under 35 U.S.C. § 102(e) over *Flood* is improper and should be reversed by the Board. The rejection of Claims 2-3, 8-9, 15, 17, and 23 under 35 U.S.C. § 103(a) as unpatentable over *Flood* in view of *Schneider* is improper and should be reversed by the Board.

I. The Prosecution of this Application

Appellant first comments on the prosecution of this Application. Appellant previously appealed similar rejections by the Examiner of Claims 1-21. In a decision issued by the Board on May 24, 2005 (the "*First Appeal Decision*"), the Board affirmed in-part and overturned in-part the Examiner's rejections of Claims 1-21. In particular, the Board upheld the Examiner's rejection of Claims 1, 2, 4-8, 10-16, and 18-21 and overturned the Examiner's rejection of Claims 3, 9, and 17.

In response to the *First Appeal Decision*, the Appellant filed a Request for Rehearing on July 22, 2005 in which Appellant noted that *Flood* failed to disclose any "communication link" between the terminal of *Flood* and the processing module. In response to the Request for Rehearing the Board issued another decision (the "*Rehearing Decision*") in which the Board took issue with the Appellant's argument noting that "appellant's argument is apparently based on appellant's view that a 'communication link' requires that that the complete path from terminal...to processor...must exist at the same time." *Rehearing Decision*, Page 2. The Board continued by noting, "[w]hether the examiner and the Board properly interpreted the term 'communication link' is a question of fact which should have been argued before the examiner and should be part of the record we are now reviewing." *Id.*, Page 3.

In response to the *Rehearing Decision*, Appellant filed a Request for Continued Examination on November 15, 2005 (the "*Request for Continued Examination*"), in which Appellant amended Claims 1, 7, 14, and 21 to more clearly convey the definition of "communication link" as used in the rejected claims. Additionally, Appellant added new Claims 22-26 that incorporated elements of Claim 3 that the Board indicated were not found

in the cited references. Despite these amendments and added claims, the Examiner rejected Claims 1-21 in an Office Action issued March 8, 2006 (the "*March 8 Office Action*"). In the *March 8 Office Action*, the Examiner essentially repeated the same arguments that Examiner had presented on appeal without addressing the newly added limitations and ignoring the Board's direct indication that, at the very least, Claims 3, 9, and 17 were allowable (see, e.g., *March 8 Office Action*, Page 7). Moreover, the Examiner completely failed to acknowledge the addition of new Claims 22-26, providing no indication as to their status or to whether they were even pending (see, e.g., the *March 8 Office Action*, Page 1).

Appellant filed a response to the *March 8 Office Action* on June 8, 2006 (the "*June 8 Response*") in which Appellant noted deficiencies in the Examiner's rejections, citing the *First Appeal Decision* where appropriate. Additionally, Appellant noted the fact that the Examiner had entirely ignored the newly-added Claims 22-26. Appellant requested reconsideration and allowance of all claims.

Appellant and the Examiner exchanged additional correspondence, culminating in the Examiner's issuance of the *Final Office Action*. In the *Final Office Action*, the Examiner again repeated the same arguments the Examiner had presented prior to the *First Appeal Decision* and prior to the amendment of Claims 1, 7, 14, and 21. Additionally, the Examiner rejected new Claims 22 and many of its dependents on the same grounds as Claim 1, ignoring the clear differences between the scope of the relevant claims. *Final Office Action*, Pages 2-3.

Thus, the Examiner has repeatedly failed to address the substance of Appellant's arguments. Appellant respectfully notes that the Examiner has failed to satisfy M.P.E.P. § 707.07(f) which requires "[w]here the applicant traverses any rejection, the examiner should, if he or she repeats the rejection, take note of the applicant's argument and *answer the substance of it*" (emphasis added). Moreover, the Examiner has also failed to properly distinguish and consider the individual elements of each claim, instead rejecting claims, such as Claims 22 and 24-26, based on the wording of other claims in the present Application, contrary to the principle that "[a]ll words in a claim must be considered in judging the patentability of that claim against the prior art." *In re Wilson*, 424 F.2d 1382, 165 U.S.P.Q. 494, 496 (C.C.P.A. 1970).

II. Grouping of Claims

Appellant has made an effort to group claims to reduce the burden on the Board, as contemplated by 37 C.F.R. § 41.37(c)(l)(vii). Where appropriate, Appellant presents arguments as to why particular claims subject to a ground of rejection are separately patentable from other claims subject to the same ground of rejection. To reduce the number of groups and thereby reduce the burden on the Board, Appellant does not argue individually every claim that recites patentable distinctions over the references cited by the Examiner, particularly in light of the clear allowability of Appellant's independent claims. The claims of each group provided below may be deemed to stand or fall together for purposes of this Appeal.

Appellant has concluded that the claims may be grouped together as follows:

With regard to the grounds of rejection identified as issue 1 above, the claims subject to those grounds of rejection may be grouped together as follows for purposes of this Appeal:

1. Group 1 may include Claims 1, 7, 14, and 21.

With regard to the ground of rejection identified as issue 2 above, the claims subject to that ground of rejection may be grouped together as follows for purposes of this Appeal:

2. Group 2 may include Claims 1, 4-7, 10-14, 16, 18-21
3. Group 3 may include Claims 22 and 24-26.

With regard to the ground of rejection identified as issue 3 above, the claims subject to that ground of rejection may be grouped together as follows for purposes of this Appeal:

4. Group 4 may include Claims 2, 8, and 15.
5. Group 5 may include Claims 3, 9, 17, and 23.

III. Issue I – Claims 1, 7, 14, and 21 Comply with 35 U.S.C. § 112, First Paragraph

A. Overview

The Examiner rejects Claims 1, 7, 14, and 21 under 35 U.S.C. § 112, first paragraph, because, according to the Examiner, "[these] claims contain subject matter as disclosed, 'wherein the communication link forms a complete path that couples at least the client to at least the particular interface card; and communication management information using the communication link' which was not described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention." *Final Office Action*, Page 2.

B. Standard

"A description as filed is presumed to be adequate, unless or until sufficient evidence or reasoning to the contrary has been presented by the examiner to rebut the presumption." *See, e.g., In re Marzocchi*, 439 F.2d 220, 224, 169 USPQ 367, 370 (CCPA 1971); M.P.E.P. § 2163.04. "The examiner, therefore, must have a reasonable basis to challenge the adequacy of the written description." M.P.E.P. § 2163.04 "The examiner has the initial burden of presenting by a preponderance of evidence why a person skilled in the art would not recognize in an applicant's disclosure a description of the invention defined by the claims." *In re Wertheim*, 541 F.2d 257, 263, 191 USPQ 90, 97 (CCPA 1976); M.P.E.P. § 2163.04. "In rejecting a claim, the examiner must set forth express findings of fact which support the lack of written description conclusion...." *Id.* "These findings should: (A) [i]dentify the claim limitation at issue; and (B) [e]stablish a prima facie case by providing reasons why a person skilled in the art at the time the application was filed would not have recognized that the inventor was in possession of the invention as claimed in view of the disclosure of the application as filed." *Id.*

"An applicant shows possession of the claimed invention by describing the claimed invention with all of its limitations using such descriptive means as words, structures, figures, diagrams, and formulas that fully set forth the claimed invention." *Lockwood v. American Airlines, Inc.*, 107 F.3d 1565, 1572, 41 USPQ2d 1961, 1966 (Fed. Cir. 1997); M.P.E.P. § 2163.02. "Possession may be shown in a variety of ways including description of an actual reduction to practice, or by showing that the invention was 'ready for patenting' such as by

the disclosure of drawings or structural chemical formulas that show that the invention was complete, or by describing distinguishing identifying characteristics sufficient to show that the applicant was in possession of the claimed invention." *See, e.g., Pfaff v. Wells Elecs., Inc.*, 525 U.S. 55, 68, 119 S.Ct. 304, 312, 48 USPQ2d 1641, 1647 (1998); *Regents of the University of California v. Eli Lilly*, 119 F.3d 1559, 1568, 43 USPQ2d 1398, 1406 (Fed. Cir. 1997); *Amgen, Inc. v. Chugai Pharmaceutical*, 927 F.2d 1200, 1206, 18 USPQ2d 1016, 1021 (Fed. Cir. 1991) (one must define a compound by 'whatever characteristics sufficiently distinguish it'); M.P.E.P. § 2163.02.

C. The rejection of Group 1 (Claims 1, 7, 14, and 21) under 35 U.S.C. § 112, first paragraph is improper

Claims 1, 7, 14, and 21 stand rejected under 35 U.S.C. § 112, first paragraph. Appellant respectfully submits that the rejection of Claims 1, 7, 14, and 21 under 35 U.S.C. § 112, first paragraph, is improper for at least several reasons.

First, to maintain a rejection under 35 U.S.C. § 112, first paragraph, an Examiner must "[e]stablish a prima facie case by providing reasons why a person skilled in the art at the time the application was filed would not have recognized that the inventor was in possession of the invention as claimed in view of the disclosure of the application as filed." Appellant respectfully notes that, here, the Examiner has concluded without explanation that:

> [These] claim(s) contains subject matter as disclosed, 'wherein the communication link forms a complete path that couples at least the client to at least the particular interface card; and communication management information using the communication link' which was not described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention
> *Final Office Action*, Page 2.

Because the Examiner fails to provide a prima facie case for rejection under 35 U.S.C. § 112, first paragraph, the Examiner fails to satisfy the Examiner's burden for rejecting Claims 1, 7, 14, and 21 on written description grounds. M.P.E.P. § 2163.04.

Second, the Application as originally filed *does* describe the relevant limitations in such a way as to reasonably convey to one skilled in the relevant art that the inventor, at the time the application was filed, had possession of the claimed invention. As noted above, "[p]ossession may be shown in a variety of ways including description of an actual reduction

to practice, or by showing that the invention was 'ready for patenting' *such as by the disclosure of drawings* or structural chemical formulas that show that the invention was complete, or *by describing distinguishing identifying characteristics sufficient* to show that the applicant was in possession of the claimed invention. M.P.E.P. § 2163.02 (emphasis added); *see also, e.g., Pfaff*, 525 U.S. at 68, 119 S.Ct. at 312, 48 USPQ2d at 1647.

For example, the present Application clearly indicates that "a communication link 22 comprises *any switched communication path that couples client 18 to an interface card 14* and communicates management information using any suitable communication protocols, standards, and/or formats." Page 9, Lines 24-27 (emphasis added). The present Application also notes, with respect to the flowchart of FIGURE 4, that:

> At step 108, switch 60 of management card 12 establishes one or more communication links 22 between port 62 and the ports 64 determined at step 106. In this respect, switch 60 couples client 18 to the appropriate interface cards 14 and/or network devices 24.
>
> Page 14, Lines 16-20.

Additionally, FIGURE 1 clearly illustrates a particular embodiment of management system 10 in which communication link 22 "forms a complete path that couples at least the client to at least [a] particular interface card" as recited by Claim 1. In particular FIGURE 1, illustrates a path completed between client 18 and the interface card 14 on the far left side of FIGURE 1. The specific language of Claims 7, 14, and 21 are similarly supported by the present Application as originally filed.

As a result, the Examiner fails to present a prima facie case for rejection under 35 U.S.C. § 112, first paragraph. Additionally, the present Application clearly supports the relevant limitations. Claims 1, 7, 14, and 21 are thus allowable for at least these reasons. Appellant respectfully requests reconsideration and allowance of Claims 1, 7, 14, and 21.

IV. Issue II – Claims 1, 4-7, 10-14, 16, 18-22, and 24-26 are patentable over *Flood*

Claims 1, 4-7, 10-14, 16, 18-22, and 24-26 stand rejected under 35 U.S.C. § 102(e) as being unpatentable over *Flood*. Appellant respectfully submits that *Flood* fails to support the anticipation rejections of these claims. Appellant respectfully submits that these rejections are therefore improper and should be reversed by the Board.

A. Standard for Demonstrating a Prima Facie Case of Anticipation

"A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference." *Verdegaal Bros. v. Union Oil Co. of California*, 2 U.S.P.Q.2d 1051, 1053 (Fed. Cir. 1987); M.P.E.P. § 2131. In addition, "[t]he identical invention must be shown in as complete detail as contained in the . . . claim." M.P.E.P. § 2131 citing *Richardson v. Suzuki Motor Co.*, 868 F.2d 1226, 1236, 9 U.S.P.Q.2d 1913, 1920 (Fed. Cir. 1989). Furthermore, "[t]he elements must be arranged as required by the claim." *In re Bond*, 910 F.2d 831, 15 U.S.P.Q.2d 1566 (Fed. Cir. 1990); M.P.E.P. § 2131.

B. The rejection of Group 2 (Claims 1, 4-7, 10-14, 16, and 18-21) under 35 U.S.C. § 102(e) is improper

Claims 1, 4-7, 10-14, 16, and 18-21 stand rejected under 35 U.S.C. § 102(e) as being anticipated by *Flood*. Appellant respectfully submits that these claims are clearly patentable over *Flood*. Appellant respectfully submits that these rejections are therefore improper and should be reversed by the Board.

Independent Claim 1, which Appellant discusses as an example, recites:

> A system for communicating management information, comprising:
> a first interface card;
> a second interface card; and
> a management card coupled to the first interface card and the second interface card, the management card operable to:
> receive a command from a client, the command identifying an interface card or a network device associated with an interface card;
> establish a communication link between the client and a particular one of the first interface card and the second interface card selected in response to the command communicated by the client, wherein the communication link forms a complete path that couples at least the client to at least the particular interface card; and
> communicate management information using the communication link.

Flood fails to recite, expressly or inherently, every limitation of Claim 1. At a minimum, *Flood* fails to disclose "a management card...operable to...establish a communication link between [a] client and a particular one of [a] first interface card and [a] second interface card..., *wherein the communication link forms a complete path that couples*

at least the client to at least the particular interface card" (emphasis added) as required by Claim 1.

As Appellant has previously noted, the Examiner has not addressed this limitation Claim 1 substantively with respect to any art at any point during examination. Moreover, *Flood* explicitly precludes the inclusion of such a limitation in the system of *Flood.* Appellant respectfully notes that the Examiner has repeatedly failed to address the substance of this argument as required by M.P.E.P. § 707.07(f).

More specifically, in denying the Appellant's request for a rehearing, the Board of Appeals indicated that:

> [A]ppellant's argument is apparently based on appellant's view that a "communication link" requires that the complete path from terminal 24 to processors 18 must exist at the same time.
> We note that the appellant's arguments with respect to this feature of the claimed invention in the briefs were based on whether there was a connection between terminal 24 and processor 18. We essentially found that terminal 24 was connected to processors 18. We now find that the fact that data in Flood is sent from terminal 24 to processors 18 is sufficient to meet the broadest reasonable interpretation of the term "communication link."

Response to Rehearing Request, Page 2.

In response, Appellant amended Claim 1 to clarify the meaning of the term "communication link" in Claim 1. As currently amended, Claim 1 discloses "[a] management card operable to . . . establish a communication link . . . *[that] forms a complete path that couples at least the client to at least the particular interface card*" (emphasis and underlining added). With this amendment, the mere fact that data could be communicated from terminal 24 to processor 18 in *Flood* is not sufficient grounds for a rejection of amended Claim 1. Consequently, as reasonably inferred from the Board's discussion, *Flood* does not disclose every element of Claim 1 as presently amended.

As Figure 3 of *Flood* and the corresponding text clearly indicate, to whatever extent information is communicated between terminal 24 and execution processors 18, the programmable controller 10 of *Flood* never "establish[es] a communication link . . . *[that] forms a complete path"* that couples terminal 24 to execution processors 18 as required by amended Claim 1. This is because at no point in time does the system of *Flood* establish all portions of a "communication link" so as to form a "complete path." While particular portions of a possible path may exist at a particular time, the remaining portions of the path

will not exist at that same particular time. As a result, a "complete path" is never formed coupling the client and one of the interface cards.

In particular, the system disclosed by *Flood* includes a terminal 24 that couples to communication buses 21-23 through line driver 52 and 54 and serial input/output controller (SIO) 48. Column 7, lines 11-16. The *Flood* system additionally includes a RAM 38 that is also located on communication buses 31-33 and that is "for temporary storage of data received from or to be sent to the various external devices connected to the system controller". Furthermore, "[a direct memory access (DMA)] circuit 42 allows the SIO 48 to access RAM 38 to store or obtain data which have been received or will be transmitted over their respective external communication channels." Column 7, lines 31-35. Thus, data communicated to system controller 10 by terminal 24 is first transmitted to RAM 38 over communication buses 31-33.

Significantly, however, to whatever extent execution processors 18 may subsequently have direct access to RAM 38, they are not able to access communication buses 31-33 while terminal 24 is communicating on communication buses 31-33. As *Flood* indicates:

> Access to the communication buses 31-33 is controlled by an arbitration circuit 40 which resolves conflicts when several devices request access to these busses at the same time. The arbitration circuit 40 determines which component of the communication section will have access to the shared buses 31-33. A device seeking the buses sends a request signal to the arbitration circuit 40 via a line of the control bus 31 and the arbitration circuit grants the request to one device at a time by producing an access signal on another control line for that device.
>
> Column 7, lines 36-46.

Thus, to whatever extent execution processors 18 directly access RAM 38 over communication buses 31-33, execution processors 18 do not communicate over communication buses 31-33 concurrently with terminal 24 and only one device may be transmitting data to or receiving data from RAM 38 at a time. As a result, no "communication link" that forms a "complete path" between terminal 24 and execution processors 18 is ever established, despite any indirect communication of information from terminal 24 to execution processors 18. Thus, *Flood* does not disclose a management card operable to "establish a communication link . . . [that] forms a complete path that couples at least the client to at least the particular interface card" as recited by amended Claim 1.

As a result, *Flood* fails to recite, expressly or inherently, every element of amended Claim 1. Amended Claim 1 is thus allowable for at least these reasons. Appellant respectfully requests reconsideration and allowance of Claim 1 and its dependents.

Although of differing scope from Claim 1, Claims 7, 14, and 21 include elements that, for reasons substantially similar to those discussed with respect to Claim 1, are not disclosed by *Flood*. Claims 7, 14, and 21 are thus allowable for at least these reasons. Appellant respectfully requests reconsideration and allowance of Claims 7, 14, and 21, and their respective dependents, including without limitation Claims 4-6, 10-13, 16, and 18-20.

D. The rejection of Group 3 (Claims 22 and 24-26) under 35 U.S.C. § 102(e) is improper

Claims 22 and 24-26 also stand rejected under 35 U.S.C. § 102(e) as being anticipated by *Flood*. Appellant respectfully submits that these claims are clearly patentable over *Flood*. Appellant respectfully submits that these rejections are therefore improper and should be reversed by the Board.

Claim 22 recites:

> A system for communicating management information, comprising:
> a first interface card coupled to a first network device that uses a first operating system;
> a second interface card coupled to a second network device that uses a second operating system; and
> a management card coupled to the first interface card and the second interface card, the management card operable to:
> receive a command from a client, the command identifying an interface card or a network device associated with an interface card;
> establish a communication link between the client and a particular one of the first interface card and the second interface card selected in response to the command communicated by the client;
> configure management information for the operating system of the network device associated with the particular interface card; and
> communicate the management information using the communication link.

Flood fails to recite either, expressly or inherently, every element of Claim 22 for at least several reasons. As discussed in greater detail below with respect to Claim 3, *Flood*, at a minimum, fails to disclose "[a] first interface card [that] is coupled to a first network device that uses a first operating system" and "[a] second interface card [that] is coupled to a second

network device that uses a second operating system." As the Board previously noted, in addressing previously-appealed Claim 3, "the examiner's 'finding' of different operating systems in Flood is nothing more than speculation and has no support in the reference." *First Appeal Decision*, Page 12. Moreover, as the Board previously agreed, "the Flood-Schneider combination [and, by extension, *Flood* alone] does not even discuss operating systems or different operating systems associated with different network devices." *Id.* Thus, *Flood* fails to recite "[a] first interface card [that] is coupled to a first network device that uses a first operating system" and "[a] second interface card [that] is coupled to a second network device that uses a second operating system."

As a result, *Flood* fails to recite, expressly or inherently, every element of Claim 22. Claim 22 is thus allowable for at least these reasons. Appellant respectfully requests allowance of Claim 22 and its dependents.

V. Issue III – Claims 2-3, 8-9, 15, 17, and 23 are patentable over *Flood*

Claims 2-3, 8-9, 15, 17, and 23 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over *Flood* in view of *Schneider*. Appellant respectfully submits that the proposed *Flood-Schneider* combination fails to support the obviousness rejections of these claims. Appellant respectfully submits that these rejections are therefore improper and should be reversed by the Board.

A. Standard for Demonstrating a *Prima Facie* Case of Obviousness

The question raised under 35 U.S.C. § 103 is whether the prior art taken as a whole would suggest the claimed invention taken as a whole to one of ordinary skill in the art at the time of the invention. *See* 35 U.S.C. § 103(a). The M.P.E.P. sets forth the strict legal standard for establishing a *prima facie* case of obviousness based on modification or combination of prior art references. "To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references where combined) must teach or suggest all the claim limitations." M.P.E.P. § 2142, 2143.

Moreover, "[t]o establish *prima facie* obviousness of a claimed invention, all the claim limitations must be taught or suggested by the prior art. All words in a claim must be considered in judging the patentability of that claim against the prior art." M.P.E.P. § 2143.03 (citations omitted).

B. The rejection of Group 4 (Claims 2, 8, and 15) under 35 U.S.C. § 103(a) is improper

Claims 2, 8, and 15 depend from Claims 1, 7, and 14 respectively which have all been shown above to be allowable over *Flood*. Combining *Flood* with *Schneider* fails to remedy the omissions of *Flood*. For example, *Schneider* also fails to disclose "a management card...operable to...establish a communication link between [a] client and a particular one of [a] first interface card and [a] second interface card..., wherein the communication link forms a complete path that couples at least the client to at least the particular interface card" (emphasis added) as required by Claim 2 (based on its dependency on Claim 1). *Schneider* additionally fails to satisfy similar limitations of Claims 8 and 15.

As a result, *Flood* and *Schenider*, both alone and in combination, fail to disclose, teach, or suggest every element of Claims 2, 8, and 15. Claims 2, 8, and 15 are thus allowable for at least these reasons. Appellant respectfully requests reconsideration and allowance of Claims 2, 8, and 15.

C. The rejection of Group 5 (Claims 3, 9, 17, and 23) under 35 U.S.C. § 103(a) is improper

For example, Claim 3 depends from Claim 2, another dependent of Claim 1. Claim 2 recites:

> The system of Claim 1, wherein the management card comprises:
> a switch operable to establish the communication link between the client and one of a first port and a second port of the management card;
> a memory operable to store mapping information associating the first port with the first interface card and the second port with the second interface card; and
> a processor coupled to the memory and the switch, the processor operable to:
> receive the command;

> determine the port associated with the particular interface card using the mapping information; and
>
> command the switch to establish the communication link between the client and the determined port.

Meanwhile Claim 3 recites:

> The system of Claim 2, wherein:
>
> the first interface card is coupled to a first network device that uses a first operating system;
>
> the second interface card is coupled to a second network device that uses a second operating system; and
>
> the processor is further operable to configure the management information for the operating system of the network device associated with the particular interface card.

The proposed *Flood-Schneider* combination fails to disclose every element of Claim 3. At a minimum, the proposed *Flood-Schneider* combination fails to disclose "[a] first interface card [that] is coupled to a first network device that uses a first operating system" and "[a] second interface card [that] is coupled to a second network device that uses a second operating system." As Appellant repeatedly noted during prosecution, the Examiner, in addressing this element, continues to assert an argument the Board has already rejected. *See, e.g., Final Office Action*, Page 7. In particular, the Examiner asserts that:

> Flood disclosed wherein: the first interface card is coupled to a first network device that uses a first operating system the second interface card is coupled to a second network device that uses a second operating system (col. 4, lines 33-49); and the processor is further operable to configure the management information for the operating system of the network device associated with the particular interface card (col. 4, lines 61-67).
>
> *Final Office Action*, Page 7.

The Board, however, previously rejected this argument on appeal.

In particular, the Board stated in its Decision on Appeal, issued May 24, 2005 (the "Board's Decision"):

> The [*Flood-Schneider* combination] makes no mention of first and second operating systems and of configuring management information for the operating system. The examiner's "finding" of different operating systems in Flood is nothing more than speculation and has no support in the reference....The prior art applied in the examiner's rejection simply does not provide the support needed to reject claims 3, 9 and 17."
>
> *First Appeal Decision*, Pages 12-13.

Thus, as the Board notes, the proposed *Flood-Schneider* fails to disclose a "first operating system" and "a second operating system" and, thus, fails to disclose "[a] first interface card [that] is coupled to a first network device that uses a first operating system" and "[a] second interface card [that] is coupled to a second network device that uses a second operating system" as recited by Claim 3. Appellant has noted this deficiency repeatedly (see, e.g., *June 8 Response*, Pages 14-15), but the Examiner has failed to address this argument. In the *Final Office Action*, the Examiner again quotes the same language that the Examiner improperly relied on in rejecting Claim 3 prior to the first appeal of the present Application, contrary to the ruling of the Board. *Final Office Action*, Page 7.

Thus, as both Appellant and the Board have noted, the proposed *Flood-Schneider* combination fails to disclose, teach, or suggest every element of Claim 3. As a result, the proposed *Flood-Schneider* combination can not properly be used to reject Claim 3. Moreover, this conclusion is in accordance with the findings of the Board. *First Appeal Decision*, Page 13. Claim 3 is thus allowable for at least this reason. Appellant respectfully requests reconsideration and allowance of Claim 3.

Although of differing scope from Claim 3, Claims 9, 17, and 23 include certain elements that, for reasons substantially similar to those discussed with respect to Claim 3, are not disclosed by the proposed *Flood-Schneider* combination. Claim 9, 17, and 23 are thus allowable for the reasons discussed with respect to Claim 3. Appellant respectfully requests reconsideration and allowance of Claims 9, 17, and 23.

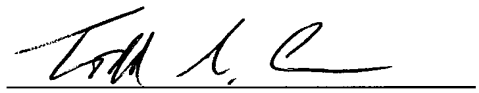
Conclusion

Appellant has demonstrated that, for at least the foregoing reasons, the present invention, as claimed, is clearly patentable over the references cited by the Examiner. Therefore, Appellant respectfully requests the Board to reverse the final rejection of the Examiner and instruct the Examiner to issue a Notice of Allowance of all pending claims.

The Commissioner is hereby authorized to charge the large entity fee of $500.00 under 37 C.F.R. §§1.191(a) and 1.17(b) for filing this Appeal Brief to Deposit Account No. 02-0384 of Baker Botts L.L.P. Although no other fees are believed to be due at this time, the Commissioner is hereby authorized to charge any additional fees and/or credit any overpayments to Deposit Account No. 02-0384 of Baker Botts L.L.P.

Respectfully submitted,

BAKER BOTTS L.L.P.
Attorneys for Appellant



Todd A. Cason
Reg. No. 54,020

Date: 5/18/07

Customer Number: **05073**

Appendix A

The Claims:

1. (Previously presented) A system for communicating management information, comprising:

a first interface card;

a second interface card; and

a management card coupled to the first interface card and the second interface card, the management card operable to:

receive a command from a client, the command identifying an interface card or a network device associated with an interface card;

establish a communication link between the client and a particular one of the first interface card and the second interface card selected in response to the command communicated by the client, wherein the communication link forms a complete path that couples at least the client to at least the particular interface card; and

communicate management information using the communication link.

2. (Previously presented) The system of Claim 1, wherein the management card comprises:

a switch operable to establish the communication link between the client and one of a first port and a second port of the management card;

a memory operable to store mapping information associating the first port with the first interface card and the second port with the second interface card; and

a processor coupled to the memory and the switch, the processor operable to:

receive the command;

determine the port associated with the particular interface card using the mapping information; and

command the switch to establish the communication link between the client and the determined port.

3. (Original) The system of Claim 2, wherein:

the first interface card is coupled to a first network device that uses a first operating system;

the second interface card is coupled to a second network device that uses a second operating system; and

the processor is further operable to configure the management information for the operating system of the network device associated with the particular interface card.

4. (Original) The system of Claim 1, wherein the communication link comprises a serial communication path.

5. (Original) The system of Claim 1, wherein the command comprises information selecting one of the first interface card and the second interface card.

6. (Original) The system of Claim 1, wherein the management information comprises information used to configure a network device associated with the particular interface card.

7. (Previously presented) A method for communicating management information performed by a management card, comprising:

receiving a command from a client, the command identifying a particular one of a first interface card and a second interface card;

establishing a communication link between the client and the particular interface card in response to receiving the command, wherein the communication link forms a complete path that couples at least the client to at least the particular interface card; and

communicating management information using the communication link.

8. (Original) The method of Claim 7, further comprising storing mapping information that associates a first port of a switch with the first interface card and a second port of the switch with the second interface card, the step of establishing a communication link comprising determining the port associated with the particular interface card using the mapping information and establishing the communication link between the client and the determined port using the switch.

9. (Original) The method of Claim 7, wherein:

the first interface card is coupled to a first network device that uses a first operating system;

the second interface card is coupled to a second network device that uses a second operating system; and

further comprising configuring the management information for the operating system of the network device associated with the particular interface card.

10. (Original) The method of Claim 7, further comprising operating the client to generate the command and the management information.

11. (Original) The method of Claim 7, wherein the communication link comprises a serial communication path.

12. (Original) The method of Claim 7, wherein the command comprises information selecting one of the first interface card and the second interface card.

13. (Original) The method of Claim 7, wherein the management information comprises information used to configure a network device associated with the particular interface card.

14. (Previously presented) A management card, comprising:

a switch coupled to a first interface card and a second interface card; and

a processor coupled to the switch and operable to:

receive a command communicated by a client, the command identifying a particular one of the first interface card and the second interface card; and

command the switch to establish a communication link between the client and the particular interface card, wherein the communication link forms a complete path that couples at least the client to at least the particular interface card.

15. (Original) The management card of Claim 14, wherein:

the switch comprises a first port coupled to the first interface card and a second port coupled to the second interface card, the switch operable to establish the communication link between a client and one of the first port and the second port;

the management card further comprises a memory coupled to the processor and operable to store mapping information that associates the first port with the first interface card and the second port with the second interface card; and

the processor is further operable to:

determine the port associated with the particular interface card using the mapping information; and

command the switch to establish the communication link between the client and the determined port.

16. (Original) The management card of Claim 14, wherein the processor is further operable to communicate management information using the communication link.

17. (Original) The management card of Claim 14, wherein:

the first interface card is coupled to a first network device that uses a first operating system;

the second interface card is coupled to a second network device that uses a second operating system; and

the processor is further operable to configure the management information for the operating system of the network device associated with the particular interface card.

18. (Original) The management card of Claim 14, wherein the communication link comprises a serial communication path.

19. (Original) The management card of Claim 14, wherein the command comprises information selecting one of the first interface card and the second interface card.

20. (Original) The management card of Claim 14, wherein the management information comprises information used to configure a network device associated with the particular interface card.

21. (Previously presented) A system for communicating management information, comprising:

means for receiving a command from a client, the command identifying a particular one of a first interface card and a second interface card;

means for establishing a communication link between the client and the particular interface card in response to receiving the command, wherein the communication link forms a complete path that couples at least the client to at least the particular interface card; and

means for communicating management information using the communication link.

22. (Previously presented) A system for communicating management information, comprising:

a first interface card coupled to a first network device that uses a first operating system;

a second interface card coupled to a second network device that uses a second operating system; and

a management card coupled to the first interface card and the second interface card, the management card operable to:

receive a command from a client, the command identifying an interface card or a network device associated with an interface card;

establish a communication link between the client and a particular one of the first interface card and the second interface card selected in response to the command communicated by the client;

configure management information for the operating system of the network device associated with the particular interface card; and

communicate the management information using the communication link.

23. (Previously presented) The system of Claim 22, wherein the management card comprises:

a switch operable to establish the communication link between the client and one of a first port and a second port of the management card;

a memory operable to store mapping information associating the first port with the first interface card and the second port with the second interface card; and

a processor coupled to the memory and the switch, the processor operable to:

receive the command;

determine the port associated with the particular interface card using the mapping information; and

command the switch to establish the communication link between the client and the determined port.

24. (Previously presented) The system of Claim 22, wherein the communication link comprises a serial communication path.

25. (Previously presented) The system of Claim 22, wherein the command comprises information selecting one of the first interface card and the second interface card.

26. (Previously presented) The system of Claim 22, wherein the management information comprises information used to configure a network device associated with the particular interface card.

Appendix B

1. U.S. Patent No. 4,937,777 issued to Flood et al. (referenced hereinabove as "*Flood*")

2. U.S. Patent No. 6,304,895 issued to Schneider et al. (referenced hereinabove as "*Schneider*")

Appendix B

Ref. 1.

United States Patent [19]

Flood et al.

[11] **Patent Number: 4,937,777**

[45] **Date of Patent: Jun. 26, 1990**

[54] **PROGRAMMABLE CONTROLLER WITH MULTIPLE TASK PROCESSORS**

[75] Inventors: **Mark A. Flood; Michael D. Kalan,** both of Mayfield Heights; **Peter N. Preis,** Lyndhurst, all of Ohio; **Alden L. Peterson,** Brooklyn, N.Y.

[73] Assignee: **Allen-Bradley Company, Inc.,** Milwaukee, Wis.

[21] Appl. No.: **105,815**

[22] Filed: **Oct. 7, 1987**

[51] Int. Cl.[5] **G05B 19/00**

[52] U.S. Cl. **364/900**

[58] Field of Search 364/900, 200

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,364,472	1/1968	Sloper .	
4,128,876	12/1978	Ames et al. .	
4,293,924	10/1981	Struger et al. .	
4,302,820	11/1981	Struger et al.	364/900
4,338,675	7/1982	Palmer et al. .	
4,404,651	9/1983	Grudowski	364/900
4,413,319	11/1983	Schultz et al. .	
4,442,504	4/1984	Dummermuth et al.	364/900
4,443,865	4/1984	Schultz et al.	364/900
4,455,621	6/1984	Pelley et al.	364/900
4,504,927	3/1985	Callan .	
4,521,871	6/1985	Galdun et al.	364/900
4,628,436	12/1986	Okamoto et al. .	
4,638,452	1/1987	Schultz et al.	364/900
4,648,064	3/1987	Morley	364/900
4,691,296	9/1987	Struger	364/900
4,771,403	9/1988	Maskovyak et al.	364/900
4,858,101	8/1989	Stewart et al.	364/131

FOREIGN PATENT DOCUMENTS

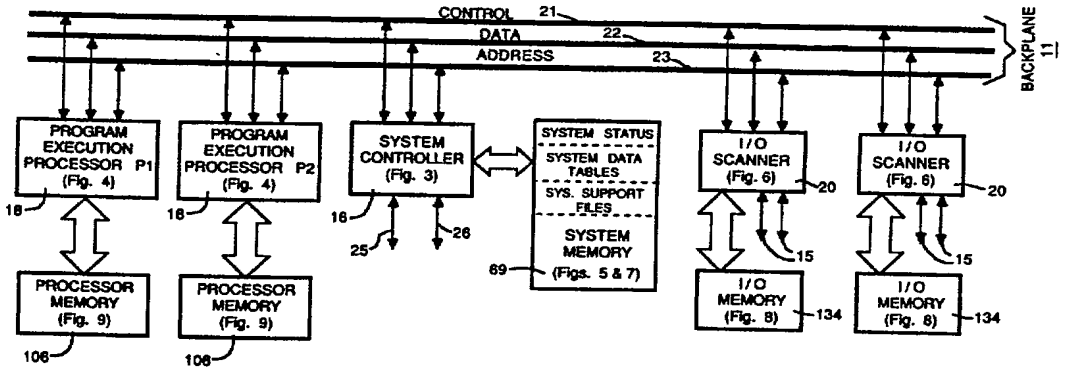
0201081	11/1986	European Pat. Off. .
2180965	4/1987	United Kingdom .

Primary Examiner—Andrew J. James
Assistant Examiner—Viet Q. Nguyen
Attorney, Agent, or Firm—Quarles & Brady

[57] **ABSTRACT**

A programmable controller for operating a machine to carry out programmed functions includes a plurality of program processors. Each of the program processors is operable to execute simultaneously a different user control program that directs the operation of the machine to perform specific functions. Each of the program processors includes a memory for storing the user control programs and function chart data. The function chart data comprises a series of descriptor files each of which contain an identification of a user control program to execute, a transition condition that indicates when the execution of that user control program is to terminate, and which descriptor file is to be processed next as well as the program processors to process it. A mechanism is also provided to enable the program processors to execute other programs in as background tasks without adversely affecting the execution of the control programs.

9 Claims, 16 Drawing Sheets



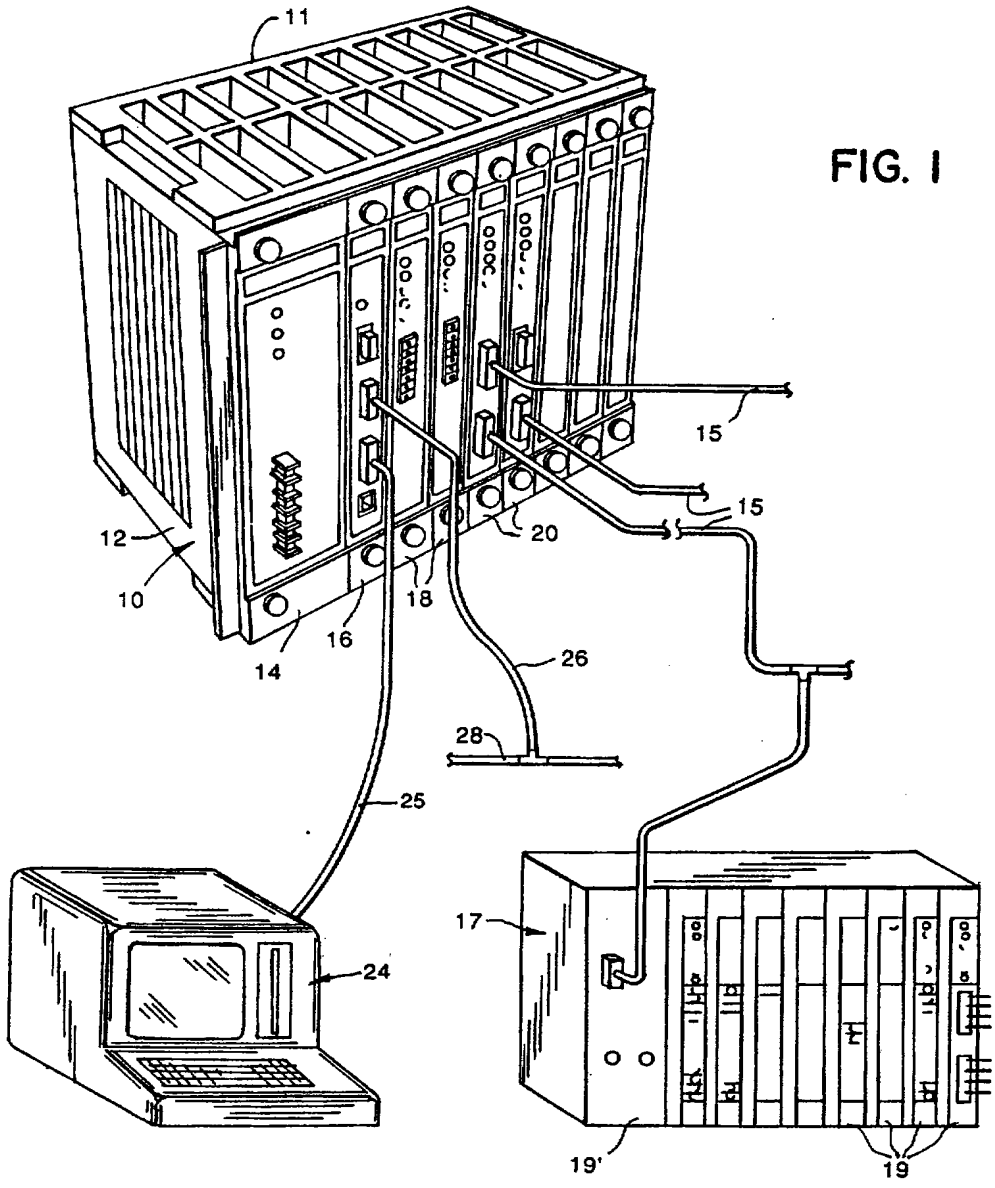


FIG. 1

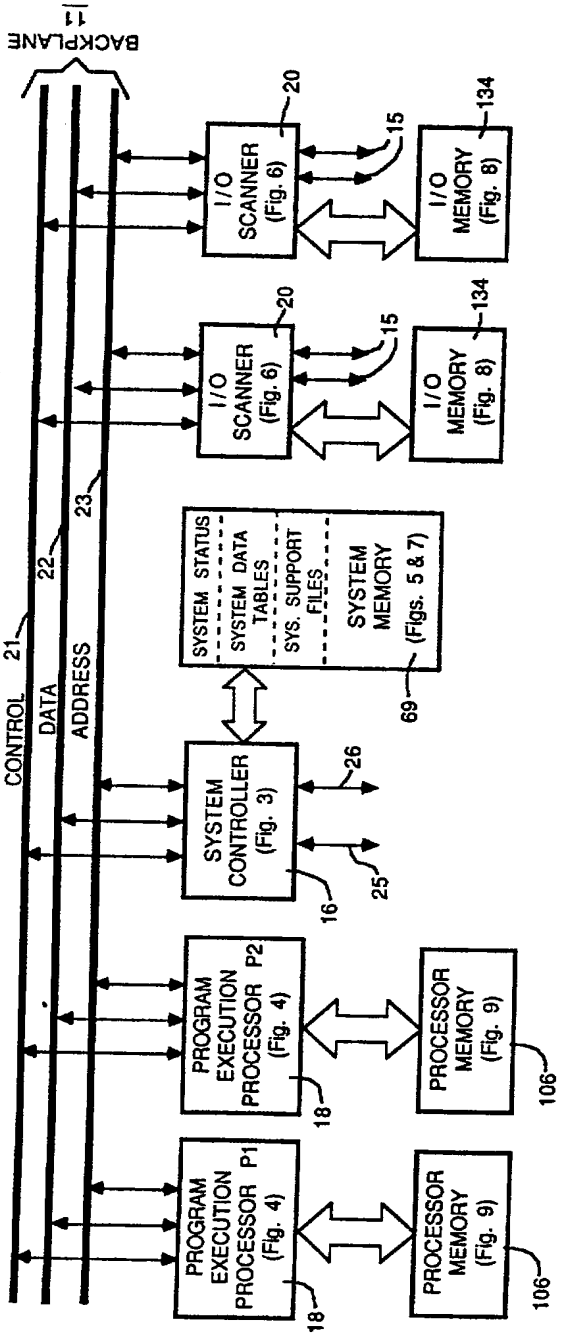

BACKPLANE
13
CONTROL
DATA
ADDRESS
21
22
23
I/O SCANNER (Fig. 6)
20
15
I/O MEMORY (Fig. 8)
134
SYSTEM STATUS
SYSTEM DATA TABLES
SYS. SUPPORT FILES
SYSTEM MEMORY (Figs. 5 & 7)
69
SYSTEM CONTROLLER (Fig. 3)
16
26
25
PROGRAM EXECUTION PROCESSOR P2 (Fig. 4)
18
PROCESSOR MEMORY (Fig. 9)
106
PROGRAM EXECUTION PROCESSOR P1 (Fig. 4)

FIG. 2

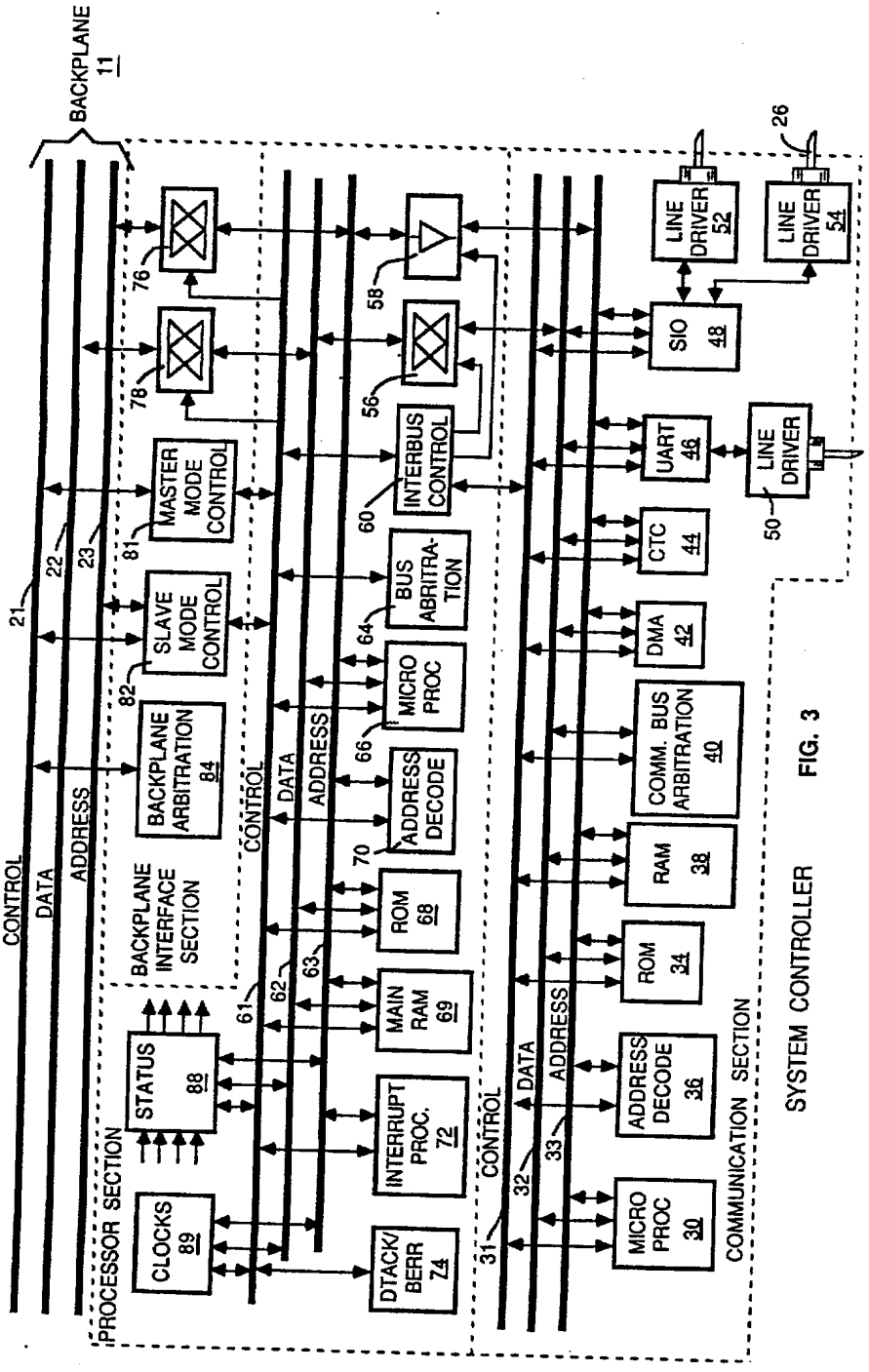

BACKPLANE 11
CONTROL
DATA
ADDRESS
BACKPLANE INTERFACE SECTION
PROCESSOR SECTION
BACKPLANE ARBITRATION 84
SLAVE MODE CONTROL
MASTER MODE CONTROL
STATUS 88
CLOCKS 89
DTACK/ BERR 74
INTERRUPT PROC. 72
MAIN RAM 69
ROM 68
ADDRESS DECODE
MICRO PROC
BUS ABRITRATION
INTERBUS CONTROL
LINE DRIVER 52
LINE DRIVER 54
LINE DRIVER
SIO 48
UART 46
CTC 44
DMA 42
COMM. BUS ARBITRATION 40
RAM 38
ROM 34
ADDRESS DECODE 36
MICRO PROC 30
COMMUNICATION SECTION
SYSTEM CONTROLLER

FIG. 3

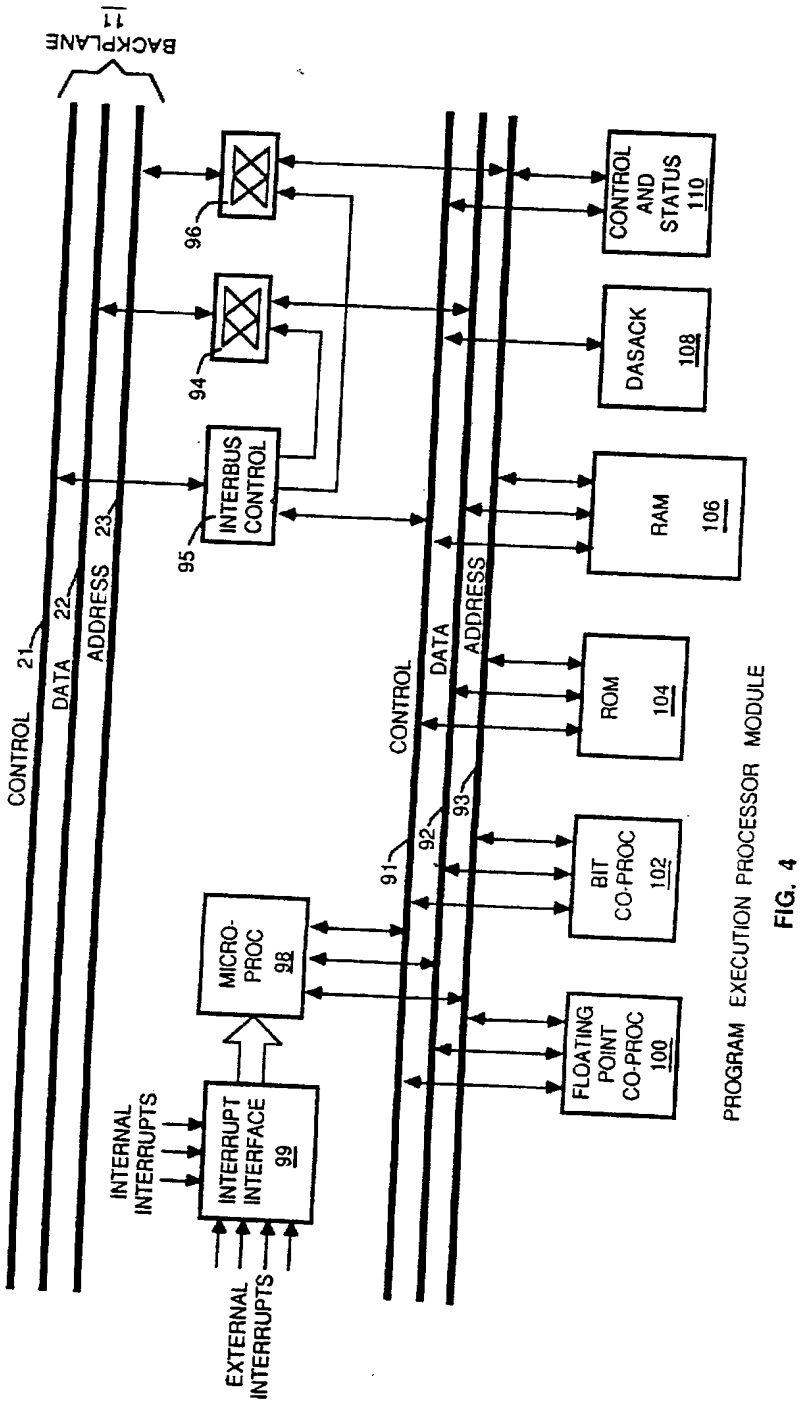


FIG. 4

PROGRAM EXECUTION PROCESSOR MODULE

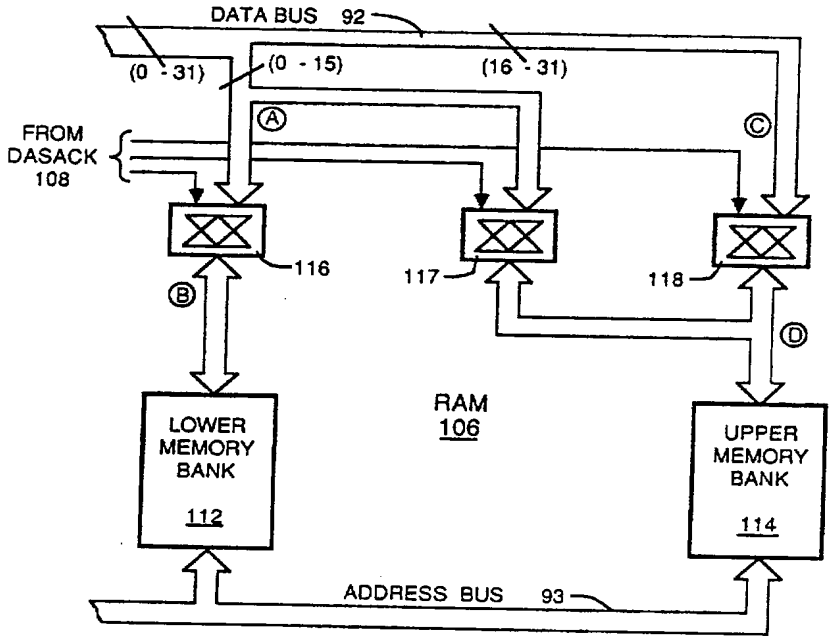
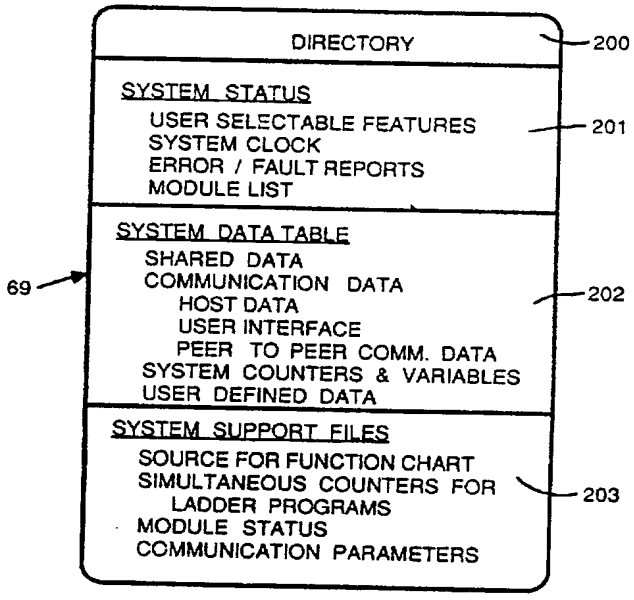
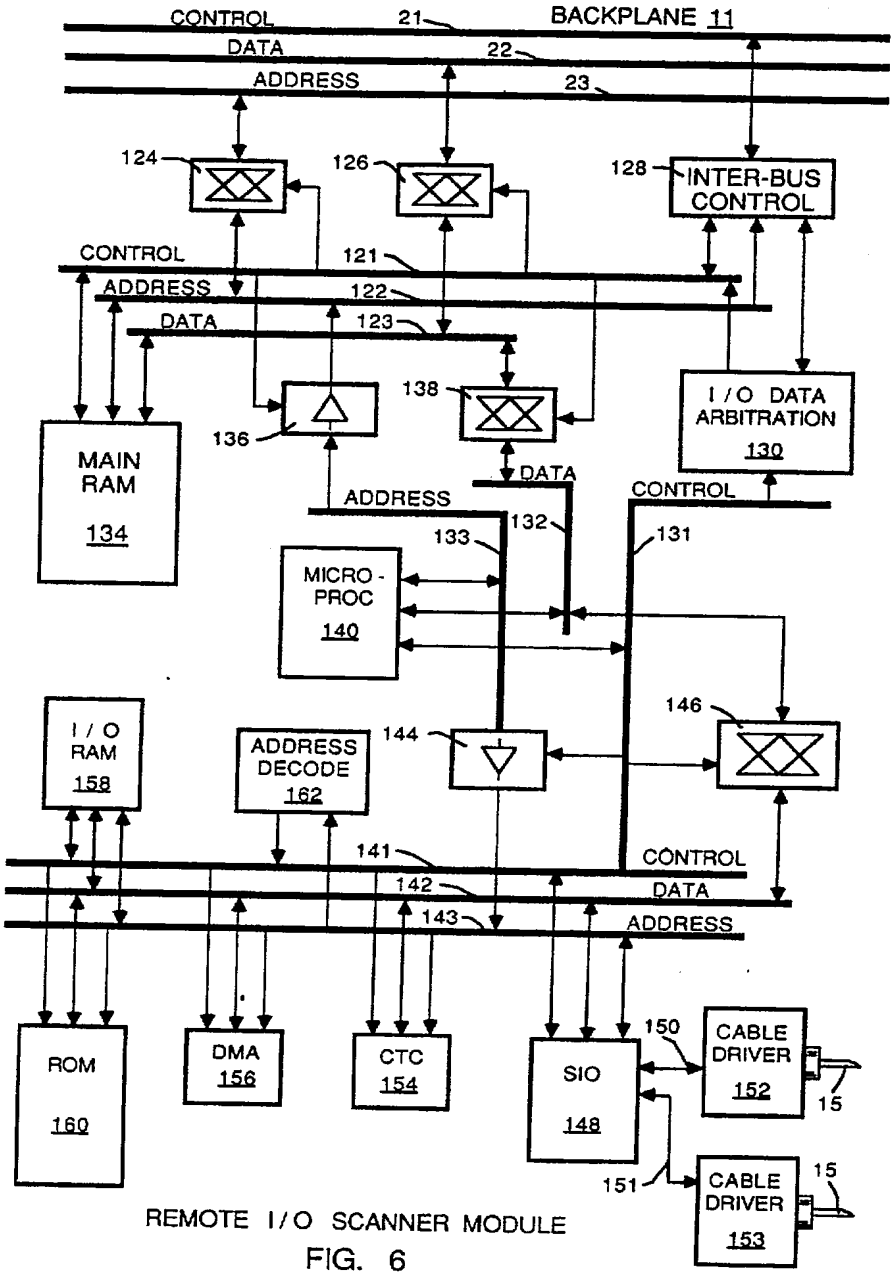


FIG. 5



SYSTEM CONTROLLER MEMORY

FIG. 7



REMOTE I/O SCANNER MODULE

FIG. 6

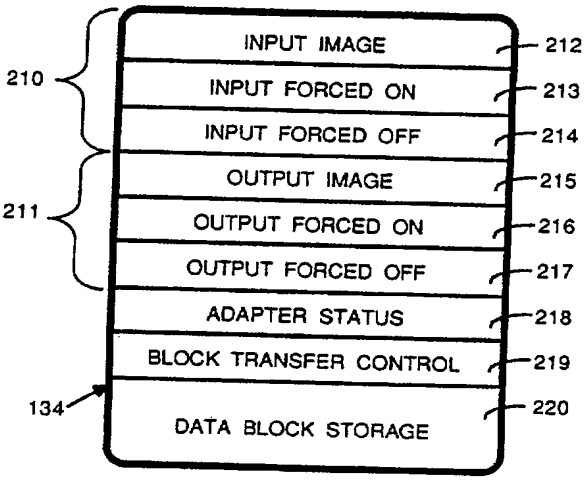

INPUT IMAGE
INPUT FORCED ON
INPUT FORCED OFF
OUTPUT IMAGE
OUTPUT FORCED ON
OUTPUT FORCED OFF
ADAPTER STATUS
BLOCK TRANSFER CONTROL
DATA BLOCK STORAGE
210
211
134
212
213
214
215
216
217
218
219
220
I / O SCANNER
MAIN RAM

FIG. 8

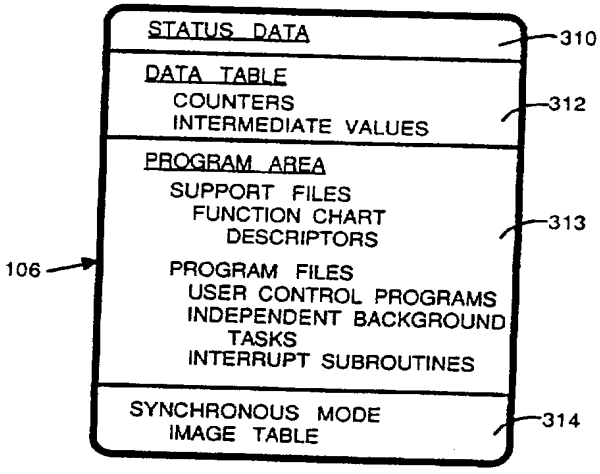

STATUS DATA
DATA TABLE
COUNTERS
INTERMEDIATE VALUES
PROGRAM AREA
SUPPORT FILES
FUNCTION CHART
DESCRIPTORS
PROGRAM FILES
USER CONTROL PROGRAMS
INDEPENDENT BACKGROUND
TASKS
INTERRUPT SUBROUTINES
SYNCHRONOUS MODE
IMAGE TABLE
106
310
312
313
314
PROGRAM EXECUTION
PROCESSOR MEMORY

FIG. 9

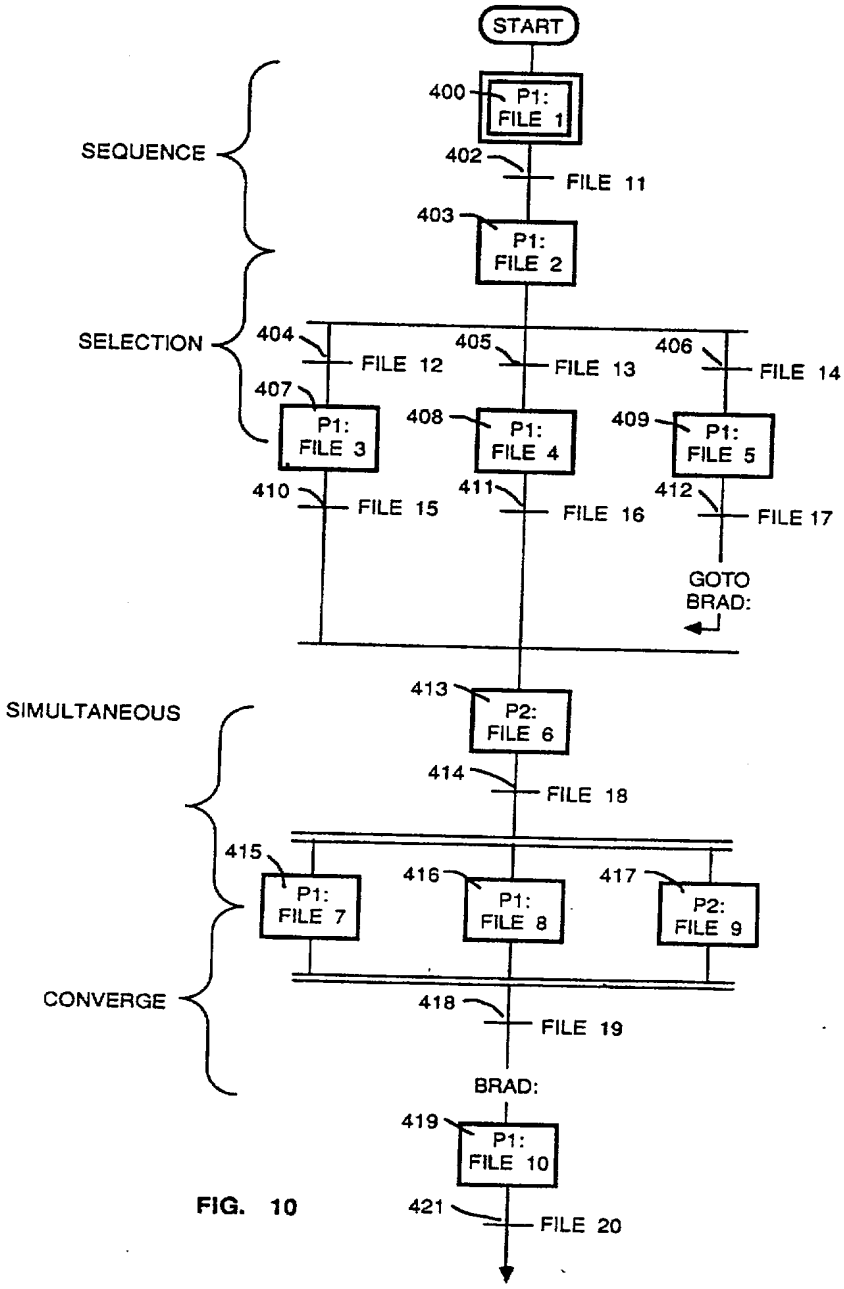

START
400
P1: FILE 1
SEQUENCE
402
FILE 11
403
P1: FILE 2
SELECTION
404
FILE 12
405
FILE 13
406
FILE 14
407
P1: FILE 3
408
P1: FILE 4
409
P1: FILE 5
410
FILE 15
411
FILE 16
412
FILE 17
GOTO BRAD:
413
P2: FILE 6
SIMULTANEOUS
414
FILE 18
415
P1: FILE 7
416
P1: FILE 8
417
P2: FILE 9
CONVERGE
418
FILE 19
BRAD:
419
P1: FILE 10
421
FILE 20

FIG. 10

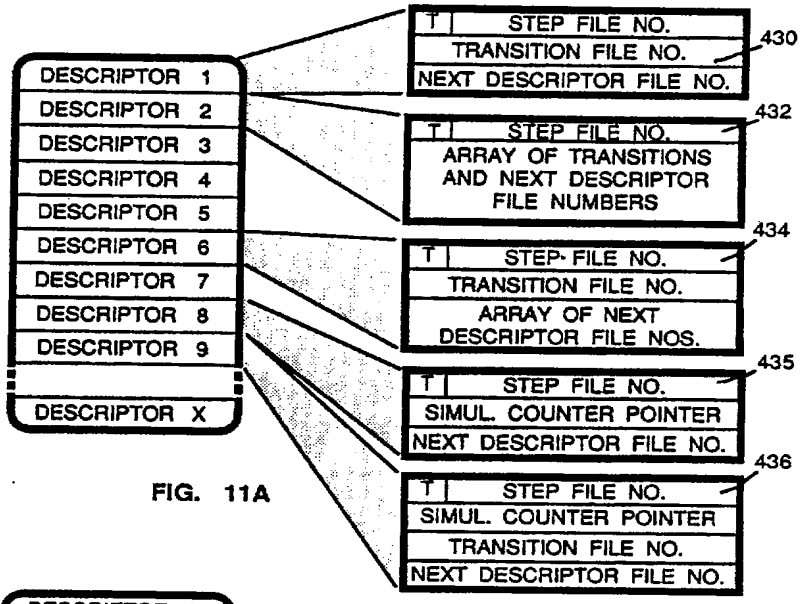
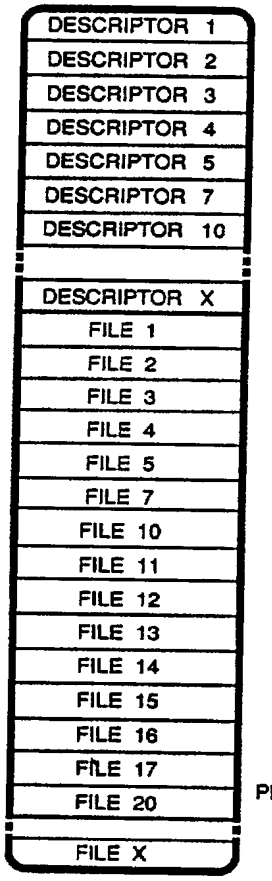
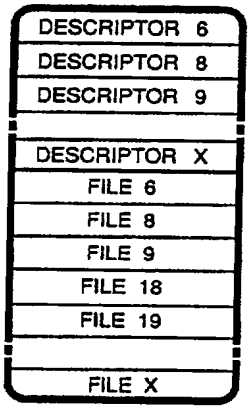


FIG. 11A



PROGRAM EXECUTION PROCESSOR MODULE P1

FIG. 11B



PROGRAM EXECUTION PROCESSOR MODULE P2

FIG. 11C

FIG. 12A

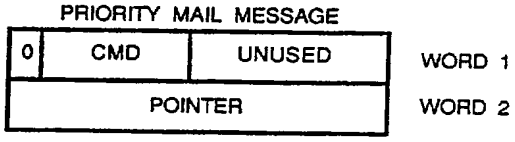
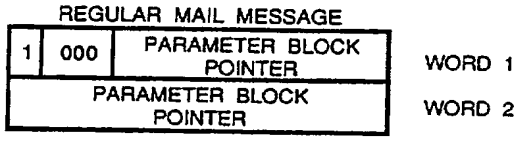
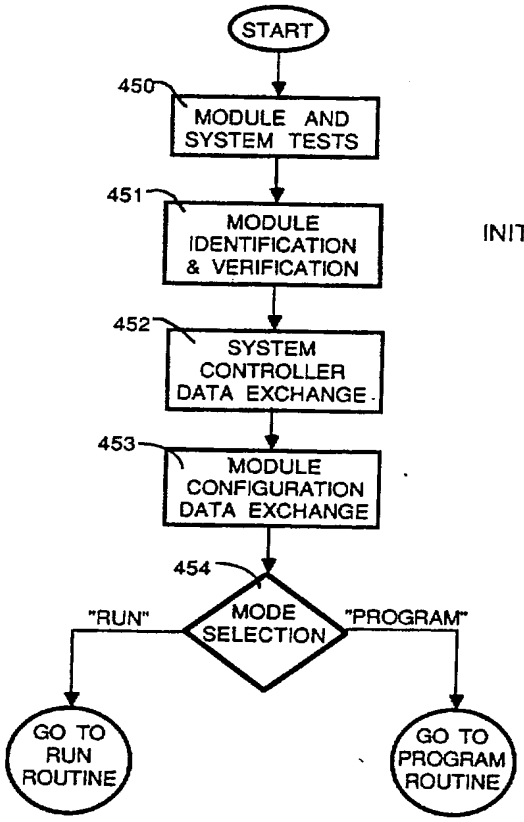


FIG. 12B





SYSTEM INITIALIZATION

FIG. 13

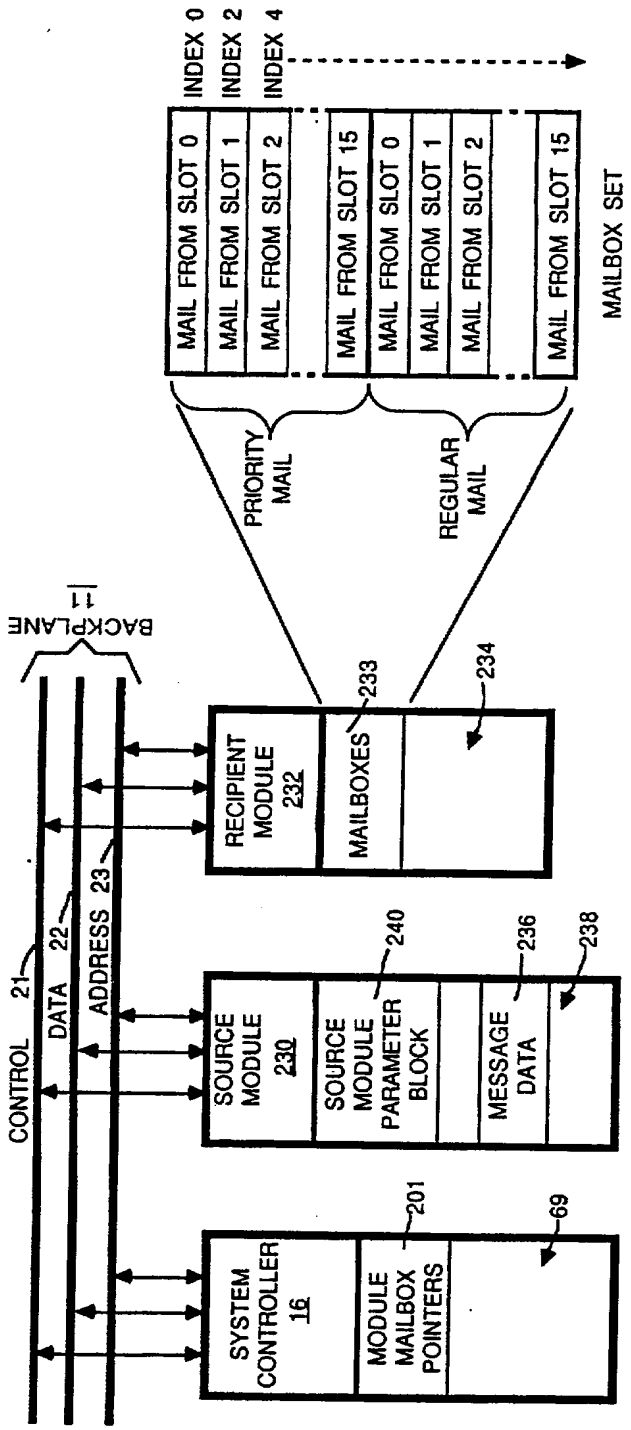
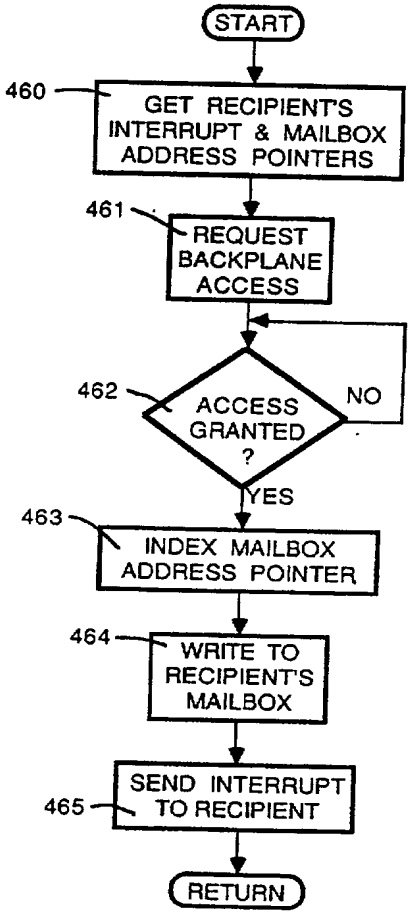

INDEX 0
INDEX 2
INDEX 4
MAIL FROM SLOT 0
MAIL FROM SLOT 1
MAIL FROM SLOT 2
MAIL FROM SLOT 15
MAIL FROM SLOT 0
MAIL FROM SLOT 1
MAIL FROM SLOT 2
MAIL FROM SLOT 15
MAILBOX SET
PRIORITY MAIL
REGULAR MAIL
11
BACKPLANE
RECIPIENT MODULE 232
MAILBOXES
233
234
CONTROL
DATA
ADDRESS
21
22
23
SOURCE MODULE 230
SOURCE MODULE PARAMETER BLOCK
240
MESSAGE DATA
236
238
SYSTEM CONTROLLER 16
MODULE MAILBOX POINTERS
201
69

FIG. 14



MAIL MESSAGE TRANSMITTAL

FIG. 15

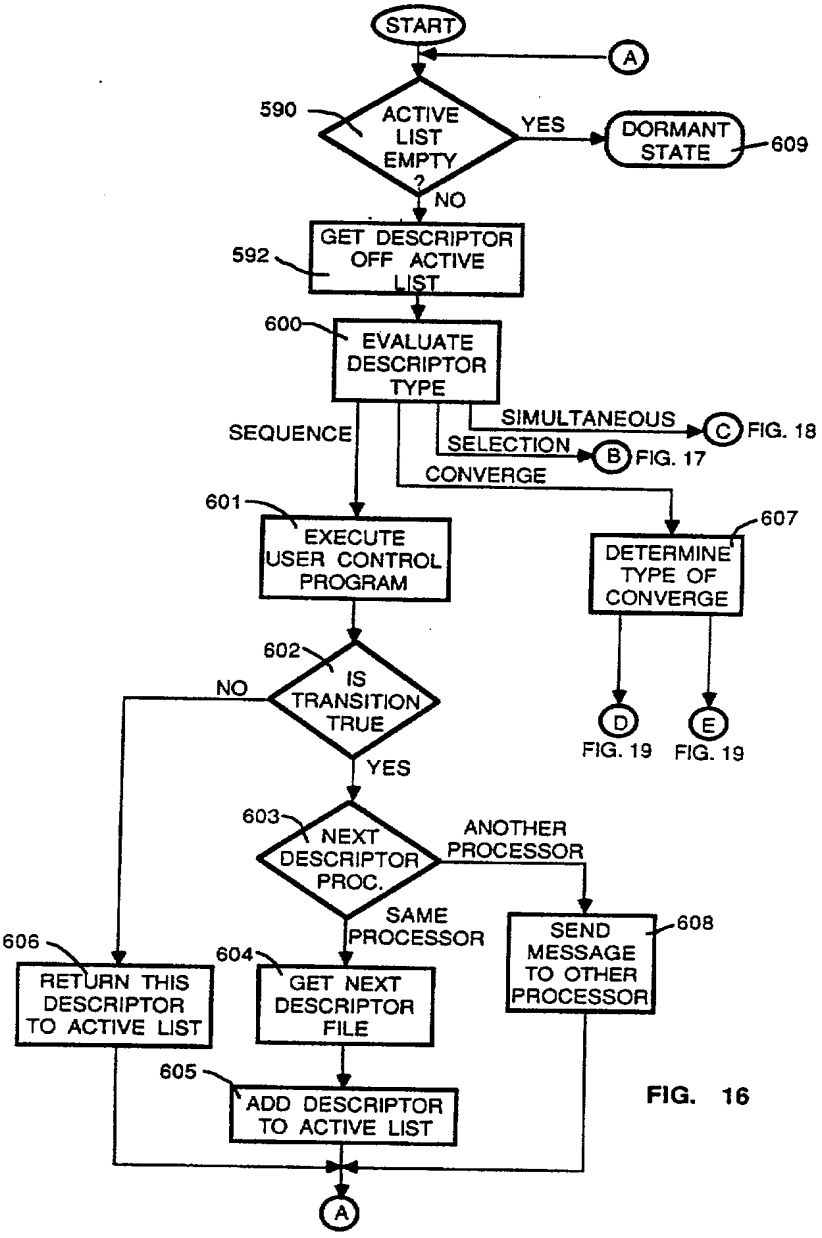
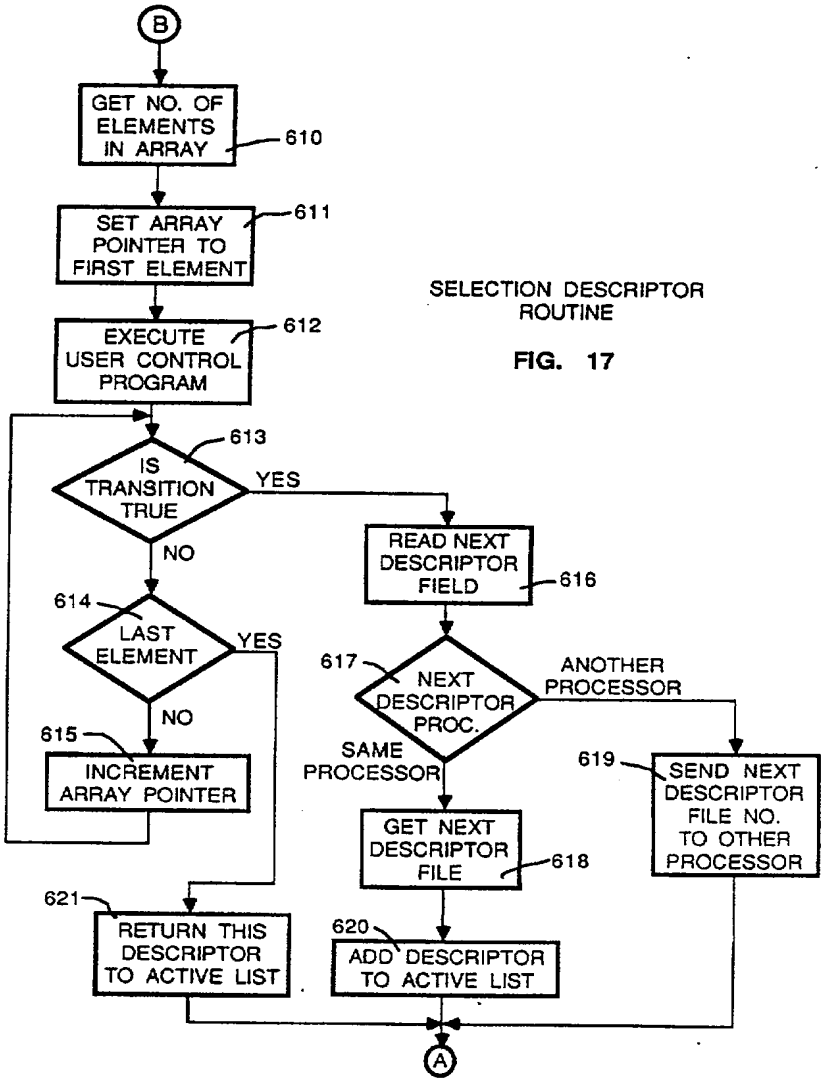
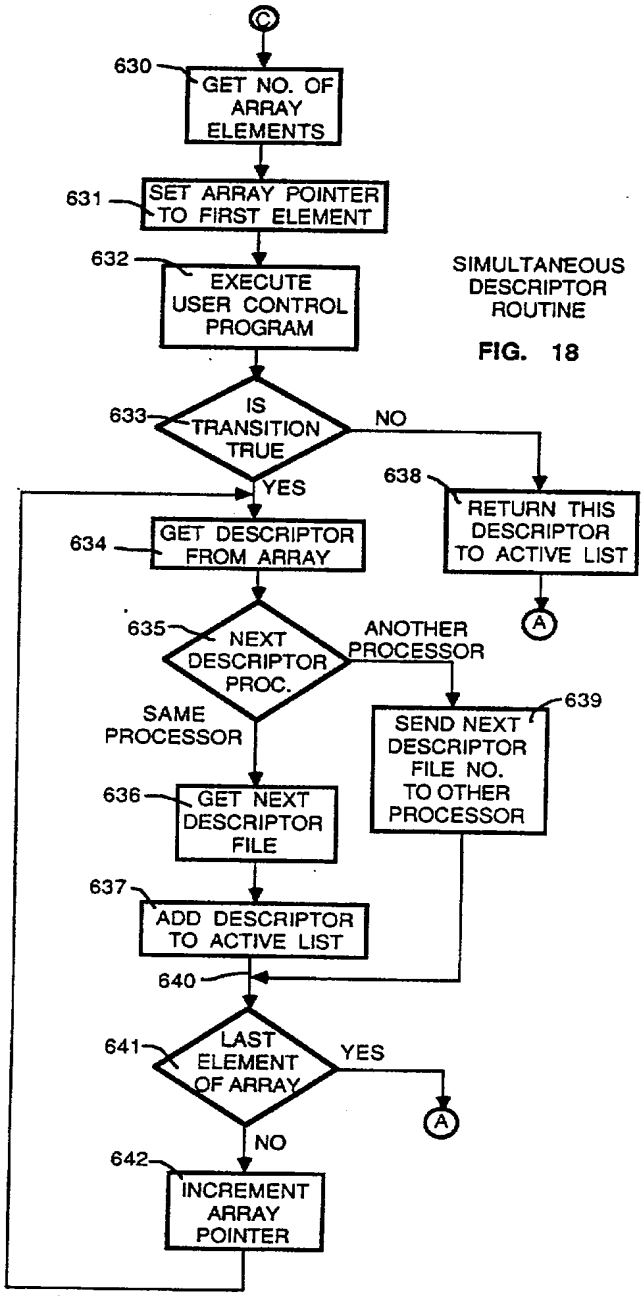

START
A
590
ACTIVE LIST EMPTY ?
YES
DORMANT STATE
609
NO
592
GET DESCRIPTOR OFF ACTIVE LIST
600
EVALUATE DESCRIPTOR TYPE
SIMULTANEOUS
C
FIG. 18
SEQUENCE
SELECTION
B
FIG. 17
CONVERGE
601
EXECUTE USER CONTROL PROGRAM
607
DETERMINE TYPE OF CONVERGE
602
IS TRANSITION TRUE
NO
YES
D
FIG. 19
E
FIG. 19
603
NEXT DESCRIPTOR PROC.
ANOTHER PROCESSOR
SAME PROCESSOR
606
RETURN THIS DESCRIPTOR TO ACTIVE LIST
604
GET NEXT DESCRIPTOR FILE
608
SEND MESSAGE TO OTHER PROCESSOR
605
ADD DESCRIPTOR TO ACTIVE LIST
A

FIG. 16



SELECTION DESCRIPTOR ROUTINE

FIG. 17



SIMULTANEOUS DESCRIPTOR ROUTINE

FIG. 18

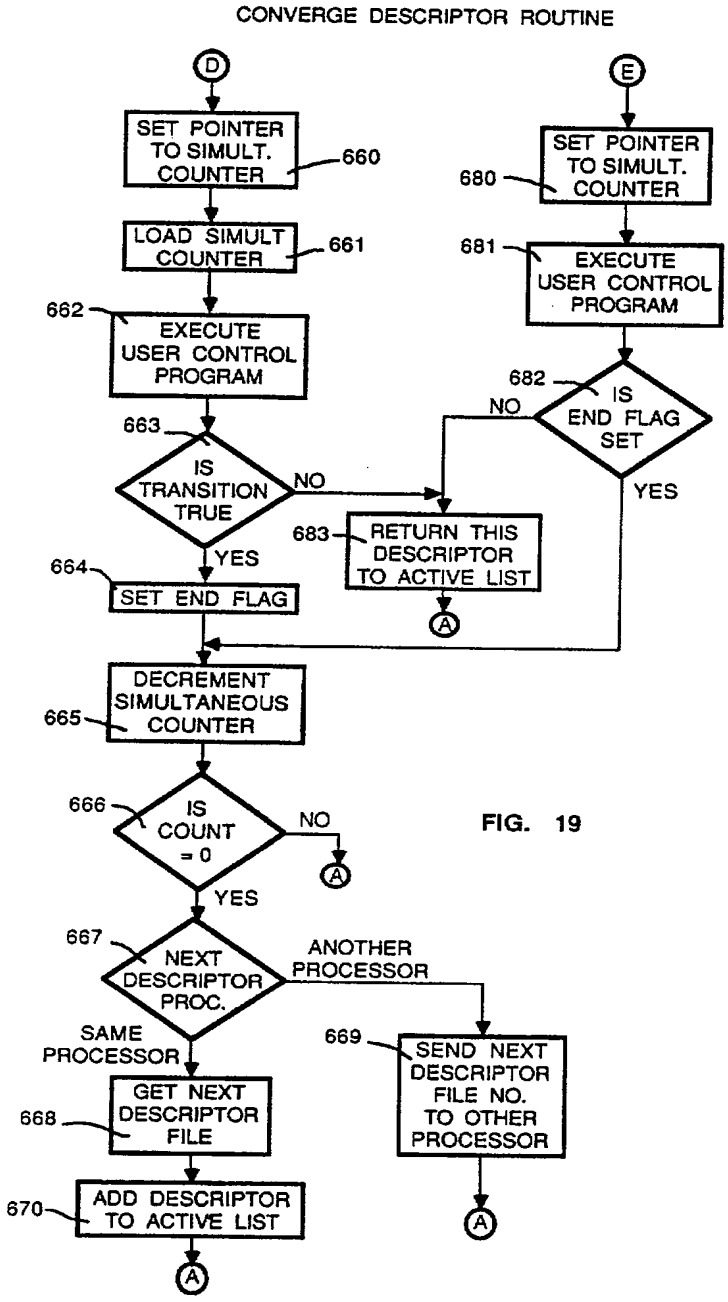

CONVERGE DESCRIPTOR ROUTINE
D
SET POINTER TO SIMULT. COUNTER
660
LOAD SIMULT COUNTER
661
662
EXECUTE USER CONTROL PROGRAM
663
IS TRANSITION TRUE
NO
YES
664
SET END FLAG
DECREMENT SIMULTANEOUS COUNTER
665
666
IS COUNT = 0
NO
YES
667
NEXT DESCRIPTOR PROC.
ANOTHER PROCESSOR
SAME PROCESSOR
668
GET NEXT DESCRIPTOR FILE
670
ADD DESCRIPTOR TO ACTIVE LIST
A
669
SEND NEXT DESCRIPTOR FILE NO. TO OTHER PROCESSOR
A
E
680
SET POINTER TO SIMULT. COUNTER
681
EXECUTE USER CONTROL PROGRAM
682
IS END FLAG SET
NO
YES
683
RETURN THIS DESCRIPTOR TO ACTIVE LIST
A

FIG. 19

PROGRAMMABLE CONTROLLER WITH MULTIPLE TASK PROCESSORS

The field of the invention is programmable controllers such as those described in U.S. Pat. Nos. 3,810,118; 3,942,158; 4,165,534; and 4,442,504.

BACKGROUND OF THE INVENTION

Programmable controllers are typically connected to industrial equipment such as assembly lines and machine tools to sequentially operate the equipment in accordance with a stored program. In programmable controllers such as those disclosed in the above cited patents, for example, the control program is stored in a memory and includes instructions which are read out in rapid sequence and executed to examine the condition of selected sensing devices on the controlled equipment, or to energize or de-energize selected operating devices on the controlled equipment contingent upon the status of one or more of the examined sensing devices.

The processor for these controllers is designed to rapidly execute programmable controller type instructions which in medium to large sized controllers includes not only instructions that manipulated single-bit input and output data, but also arithmetic instructions, file handling instructions, timers and counters, sequencers and other, more complex instructions. Such instructions have become quite standardized in the industry and they may be directly associated with elements of a ladder diagram which is easily understood by control engineers. Program panels such as those disclosed in U.S. Pat. Nos. 3,798,612 and 3,813,649 and in U.S. Pat. No. 4,070,702 have been developed to assist the user in developing and editing ladder diagram type control programs comprised of such programmable controller instructions.

To insure that the programmable controller can respond quickly to change in the status of sensing devices on the controlled system, it is imperative that the controller execute the control program repeatedly at a very high rate. The rate at which a programmable controller can execute the instructions in its instruction set, as well as the size of the control program, are the primary factors which determine the rate at which the programmable controller can repeatedly execute, or "scan", the control program.

While ladder diagram control programs are particularly easy to create and edit for relatively small to medium scale control tasks, they become cumbersome and inefficient to use in large control tasks. Large ladder diagram control programs are difficult to understand, difficult to trouble shoot, and require a long time to execute.

U.S. patent application Ser. No. 06/717,221, filed on Mar. 28, 1985 entitled "Programmable Controller with Function Chart Interpreter" addresses this problem. The controller described therein includes a processor which stores a plurality of separate ladder control programs that are logically related to each other by a stored function chart program, and the processor is operable to execute the stored function chart program which directs which ones of the stored ladder programs are to be repeatedly executed by the processor at any point in time. It has been discovered that large control tasks can usually be broken down into separate control steps which are executed in a sequential order as the controlled machine or process advances through its states. Each control step is defined by a separately executable ladder program which is easy to understand and which may be executed at a very high scan rate. The sequence in which the separate control steps are executed is defined by the function chart program which is a general expression of how the controlled machine or process is to operate. The user may thus define the general manner in which the machine or process is to operate using function chart constructs, and then define the detailed operation of the machine or process in separate, easily managed ladder programs.

The previous applications of function chart and ladder programs have been in programmable controllers having a single processor for executing the programs. Recently a programmable controller has been conceived which employs several processors each capable of simultaneously executing a separate ladder program. This type of controller requires a mechanism for coordinating the ladder programs being executed by the different processors.

Another problem developed when previous programmable controllers had to execute complex routines such as solving an equation. Such computational tasks often tied up the processor for relatively long periods of time delaying the execution of other rungs of the ladder program. This could result in certain portions of the controlled equipment not being supervised frequently enough by the programmable controller.

SUMMARY OF THE INVENTION

A programmable controller executes a single machine operation program enabling a machine to carry out a plurality of programmed functions. Each step of the machine operation program is defined as a separate user control program. The programmable controller includes a plurality of individual processor modules each of which is capable of simultaneously executing a user defined control program. A storage means is provided to store the control programs and data indicating the sequence in which the control programs are to be executed. The programmable controller also includes one or more input/output (I/O) modules for interfacing it to sensors and actuators on the controlled machine. These I/O modules may interface the controller to the sensors and actuators directly or indirectly through remote I/O adapters.

Typically, the operation of the controller is defined by a user provided function chart and a series of control programs, such as ladder programs. The function chart sets out the overall process to be performed in a sequence of steps. The function chart is broken down into a series of descriptors each of which contains data specifying a processing step to be performed, a transition condition that occurs when the step is completed, and a pointer to the next descriptor in the series. The descriptors of the function chart guide the overall flow of program execution. The programmable controller includes means for interpreting the descriptors to control the execution of the control programs by the plurality of processors.

Each processing step is assigned to one of the processor modules for execution. The step is further defined by a process control program which is either a conventional ladder diagram program or a high level language program. In the preferred embodiment, the control program is compiled and stored in a local memory of the respective processor module that has been assigned to execute the program. Once a processor module be-

gins executing a specific control program, it repeatedly carries out the execution until the associated transition condition occurs. At that time the next descriptor pointer is read for information as to the next step to perform.

The programmable controller further includes a mechanism for allotting a portion of the program execution time of each processor module for the execution of programs other than control programs. This allows complex time consuming tasks to be executed in a time slice manner without adversely interfering with the execution of the control programs.

In an embodiment of the present invention, each of the processor modules has an external interrupt which enables an external device to be coupled directly to the processor module. This external device is able to interrupt the regular program execution by the processor module and cause an interrupt routine to be executed.

A general object of the present invention is to provide a programmable controller which employs several processors to control the operation of a machine.

An object of the present invention is to provide a programmable controller wherein portions of a single machine operation program are executed simultaneously on several parallel processors.

Another object is to incorporate a mechanism in the programmable controller which directs the processor's execution of a plurality of control programs.

A further object of the present invention is to periodically execute other time consuming programs by the processor means without such execution having a substantial adverse affect on the execution of control programs.

BRIEF DESCRIPTION OF THE DRAWINGS

In the drawings which illustrate the embodiments of the present invention:

FIG. 1 is a perspective view of a programmable controller which employs the present invention;

FIG. 2 is a schematic block diagram of the programmable controller system shown in FIG. 1;

FIG. 3 is a schematic block diagram of the System Controller for the programmable controller of FIG. 2;

FIG. 4 is a schematic block diagram of a Program Execution Processor of the programmable controller shown in FIG. 2;

FIG. 5 is a schematic block diagram of the random access memory of the Program Execution Processor of FIG. 4;

FIG. 6 is a schematic block diagram of the remote input/output scanner of the FIG. 2 programmable controller;

FIG. 7 is a diagram of the System Controller memory data structure;

FIG. 8 is a diagram of the I/O Scanner memory data structure;

FIG. 9 is a diagram of the Program Execution Processor memory data structure;

FIG. 10 is an exemplary function chart diagram;

FIGS. 11 A, B and C are illustrations of the descriptor file data structures generated from the function chart program of FIG. 10;

FIGS. 12 A and B show the entries in the mailboxes of FIG. 14 for different types of messages;

FIG. 13 is a flow chart of the programmable controller initialization routine;

FIG. 14 is a schematic representation of a portion of the system of FIG. 1 used to described the communication of messages between modules of the programmable controller;

FIG. 15 depicts a flow chart of the program steps to initiate the sending of a message from one module to another in the programmable controller; and

FIGS. 16, 17, 18 and 19 are flow charts of portions of the software for interpreting function chart data and executing user control programs.

DETAILED DESCRIPTION OF THE INVENTION

With initial reference to FIG. 1, a programmable controller 10 of the present invention is housed in a rack 12 which includes a series of slots that receive a plurality of printed circuit board modules. These functional modules connect to a mother board which extends along the back surface of the rack 12 to provide a backplane 11. The backplane 11 has a plurality of module connectors which are interconnected by a conductive pattern on the backplane. The backplane 11 provides a series of signal buses to which the modules connect. The rack 12 contains a power supply module 14, a system controller 16, a number of program execution processor modules 18 and a plurality of remote input/output (I/O) scanner modules 20, although only one scanner module is required. The remaining locations in rack 12 are empty and the slots are covered by blank plates until additional functional modules are to be inserted in these slots. The physical construction of the rack 12 is disclosed in U.S. patent application Ser. No. 06/909,710 filed on Sept. 22, 1986, and assigned to the same assignee as the present invention.

Up to four remote I/O scanner modules 20 interface the controller 10 to external remote I/O racks 17 via serial I/O data links, such as link 15. Each remote I/O rack 17 has a plurality of local I/O modules 19 which are coupled to individual sensors and actuators on the controlled equipment. The local I/O modules 19 may take many forms and may include, for example, D.C. inputs or outputs, A.C. inputs or outputs, analog inputs or outputs, and open or closed loop positioning modules. The I/O racks 17 and networks 15 employ conventional interface and communication technology. The remote I/O rack 17 also contains an adapter module 19', such as the one described in U.S. Pat. No. 4,413,319, which controls the transmission of data via the I/O network 15 between the I/O modules 19 and the scanner modules 20.

The system controller 16 is connected through cable 25 to a programming terminal 24, which is used to load the user programs into the programmable controller and configure its operation, as well as monitor its performance. The terminal 24 is a personal computer programmed to enable the user to develop the control programs on the terminal, which programs are then downloaded into the programmable controller. Once the programs have been loaded into the programmable controller 10 and its operation debugged, the terminal 24 may be disconnected from the system controller 16 if further monitoring is not required. The system controller 16 may be also connected via a cable 26 to a local area network 28 over which it may receive data and programming instructions, as well as issue status information and report data to a host computer. This enables a central host computer or central terminal to program and control the operation of a plurality of programmable controllers on a factory floor.

Different steps of a function chart program are assigned to various ones of the program execution modules 18. The user control program for each step is stored in the local memory of the corresponding program execution module 18. For example the user control program may be a conventional ladder program. Several user control programs may be executed simultaneously on different ones of the program execution modules. At other times a "background task" may be executed on one program execution module 18 while another module 18 executes a user control program.

During the course of carrying out a user control program, the program execution module 18 reads input status data from the input image tables in one or more of the I/O scanner modules 20. As called for by the program instructions, the program execution module also writes output state data to the output image table in the I/O scanning module 20 that services the respective output device. Access to the I/O tables is obtained via the rack backplane 11.

When a program execution module completes a function chart step, it sends a command to the program execution module 18 containing the next step to be executed. The command identifies the next step and instructs that program execution module 18 to begin executing it.

Hardware

In order for the data and commands to be transfered among the modules of the programmable controller, the modules are interconnected as shown in FIG. 2.

Each of the blocks in FIG. 2 contains a reference to other Figures of the drawings that contain the details of the component represented by the block. Each of the modules is connected to the rack backplane 11 which consists of separate control, data and address buses, 21–23 respectively. The control bus 21 consists of a number of separate lines to which a module may connect to depending upon the control signals required for that type of module. The data bus 22 is thirty-two bits wide and the address bus is 27 bits wide.

In the preferred embodiment, each of the system controller 16, program execution processor modules 18 and remote I/O scanner modules 20 includes a removable daughter board containing a local memory for data and operating instructions for that module. The daughter board contains a battery to sustain the memory when power is removed from the controller. The memory 134 in each remote I/O scanner module contains an I/O image table that indicates the state of each of the sensor devices and the desired state of each of the controlled actuators connected to the scanner module 20. The system controller's memory 69 contains various segments that store data regarding the system status and configuration as well as information relating to specific system operations. Each of the processor modules 18 have a random access memory 106 which stores the function chart data, user control programs and their operating variables.

Each of the functional modules of the programmable controller 10 will be described in detail in the following sections.

System Controller

As noted previously, the system controller module 16 provides a communication interface for the programmable controller to external terminals and local area networks. The system controller 16 also performs system housekeeping functions, such as providing an indication of the system status and supervising access to the backplane 11.

During normal operation of the programmable controller, the system controller takes care of communication with the external devices that are connected to it, such as network 28 and programming terminal 24. The most significant task is communicating with the terminal 24 to provide information allowing the operator to monitor the system performance and to detect faulty sensors or actuators. Another task supervised by the system controller is the exchange of data with a host computer or a peer programmable controller via the local area network 28. This enables the host computer to collect statistics from one or a number of programmable controllers regarding their operation. In addition to these functions another function the system controller 16 receives all programming changes and sees to it that the program in the corresponding program execution module 18 is updated. For example, this includes adding, deleting and changing various rungs of the ladder program.

The system controller as shown schematically in FIG. 3 connects to the backplane buses 21–23 and is divided into three sections (delineated by dashed lines) for backplane interface, processing and communication operations. The backplane interface section supervises the backplane access for all the rack modules and interfaces the controller module 16 to the backplane 11. The processor section executes a supervisory program for the controller 10. The communication section is primarily responsible for communicating with external terminal 24 and local area networks, such as LAN 28. Each of the processor and communication sections includes a set of internal buses, communication buses 31–33 and processor buses 61–63 respectively.

Various circuits connected to the communication buses control the interfacing of the system controller 16 to the programming terminal 24 and the local area network 28. The communication buses consist of control bus 31 having a number of individual control lines running between various components in the communication section, an eight bit wide data bus 32 and a sixteen bit wide address bus 33. The communication section is built around a microprocessor 30, such as the model Z80 manufactured by Zilog, Incorporated. The microprocessor 30 executes machine language instructions which are stored in a read-only memory (ROM) 34. The instructions are fetched from the ROM, decoded and then executed by the microprocessor 30 to carry out the communication functions. The program controlling these functions is similar to that employed in previous programmable controllers.

A conventional address decoding circuit 36 receives each address issued by the microprocessor 30 and decodes it to produce the proper set of signals on control lines 31. For example, if the microprocessor 30 is accessing the ROM 34, the address decode circuit 36 will recognize that the address sent by the microprocessor 30 on bus 33 is within the range of addresses at which the ROM is located. Once it has recognized which device in the communications section is to be accessed, the address decode circuit 36 produces control signals for the device to carry out the access.

Two serial input/output devices, UART 46 and serial input/output controller (SIO) 48, are also connected to the three communication buses 31–33. The UART 46 may be any of several commercially available universal

asynchronous receiver/transmitter integrated circuits. The UART 46 converts the parallel data which is present on the communication data bus 32 into a properly formatted serial signal which is fed to an input/output line driver/receiver 50. The line driver 50 provides output signals corresponding to any one of several serial signal standards, such as RS232, RS423 or RS422. The serial I/O communication controller 48 may be any of several commercially available integrated circuits which service two synchronous serial communication channels. The SIO 48 interfaces the communication section of the system controller 16 to local area networks connected to the line drivers 52 and 54, such as network 28 of FIG. 1. The programming terminal 24, shown in FIG. 1, is connected to one of these line driver 52 or 54.

Also located within the communication section is a random access memory (RAM) 38 for temporary storage of data received from or to be sent to the various external devices connected to the system controller. The RAM 38 may be accessed via address bus 33 so that data may be written into or read from the memory via bus 32 depending upon enabling signals from control bus 31. RAM 38 incorporates a parity circuit which analyzes each digital byte being stored in the RAM and produces a parity bit using conventional techniques. This parity bit is employed to check the integrity of the data read from the random access memory 38. A direct memory access (DMA) circuit 42 is provided to enable rapid data exchange between the SIO 48 and the RAM 38 during the communication process. The DMA circuit 42 allows the SIO 48 to access RAM 38 to store or obtain data which have been received or will be transmitted over their respective external communication channels.

Access to the communication buses 31–33 is controlled by an arbitration circuit 40 which resolves conflicts when several devices request access to these buses at the same time. The arbitration circuit 40 determines which component of the communication section will have access to the shared buses 31–33. A device seeking the buses sends a request signal to the arbitration circuit 40 via a line of the control bus 31 and the arbitration circuit grants the request to one device at a time by producing an access signal on another control line for that device.

A counter/timer circuit (CTC) 44 connects to the communication buses 31–33 and to an interrupt terminal on microprocessor 30 in order to process interrupt requests from the other components within the communications section. The CTC 44 is also configured as a timer to produce an interrupt request to the microprocessor 30 at a given periodic interval, such as every 10 milliseconds, so that various routines may be periodically executed regardless of the task then being performed by microprocessor 30. In response to an interrupt request from the CTC 44, the microprocessor 30 reads a vector from the CTC 44 directing the microprocessor to-the appropriate interrupt service routine stored in ROM 34, such as performing a data I/O request from either UART 46 or SIO 48.

Referring still to FIG. 3, the processor section is linked together by a set of buses that comprise control lines 61, a sixteen bit data bus 62 and a twenty-three bit address bus 63. Access to these buses is controlled by an arbitration circuit 64 similar to circuit 40 on the communication buses. Two sets of data gates 56 and 58 extend between the communciation buses 32 and 33 and processor buses 62 and 63 of the system controller module 16. Specifically, the first set of gates 56 provides an eight bit bidirectional connection of the communication section data bus 32 to the data bus 62 of the processor section; and the second set of gates 58 connects the two address buses 33 and 63. The data gate 56 consists of two sets of eight individual tristate data gates, each set controlling data flow in one direction between the two buses 32 and 62. Only the lower eight lines of the processor data bus 62 are coupled to the eight bit communication section data bus 32. As addressing will only occur from the processor section to the communication section, address gates 58 consist of one set of sixteen tri-state signal gates coupling the sixteen communication address bus lines 33 to the lower sixteen address lines in the processor section. An interbus control circuit 60 is connected to control lines 61 and 31 from the processor and the communication sections, respectively, and in response to access request signals from arbitration circuits 40 and 64, the control circuit 60 enables the data and address buffers 56 and 58.

The processor section is built around a sixteen-bit microprocessor 66, such as a model 68010 manufactured by Motorola Inc., which executes program code stored in read only memory 68. The 68010 microprocessor is essentially a memory mapped device and does not have any input/output lines directly connected to it. Therefore, its access to other components on the processor bus must be accomplished through issuing addresses on bus 63. The address sent from the microprocessor 66 is decoded in an address decode circuit 70 to produce the proper control signals for the accessed component. The processor address decoder 70 functions in much the same manner as the communication section address decoder circuit 36. The processor section also contains an interrupt processor 72 which controls interrupts to the microprocessor 66 and provides the proper instruction address vectors.

A data transfer acknowledge and bus error (DTACK/BERR) circuit 74 is also connected to the processor control bus 61. Circuit 74 responds to signals from the various components in the processor section to acknowledge the completion of a data transfer and issue bus error signals in the event of improper addressing or failure of data transfer. These signals are acted on by the microprocessor 66 which takes corrective action. The processor section also includes clock circuit 89 that contains the main system clock and a real time clock. A system status circuit 88 receives input signals related to the status of the entire system 10 and provides an indication of that status.

The main random access memory (RAM) 69 for the system controller 16 is also connected to the processor buses 61–63. The RAM 69 is a static memory containing 393,216 (384K) memory locations each of which is 16 bits wide and serves as the system memory for the entire controller 10. The system memory 69 can be directly accessed via the backplane 11 by other modules in the system without the intervention of the central processing unit 66 within the system controller.

FIG. 7 illustrates the data structures within the main system memory 69 included in the system controller module 16. The system memory 69 stores separate data files, which contain data for performing specific functions during the operation of the programmable controller. The data structures include various forms of data such as integers, floating point numbers, ASCII characters and various control structures. The first file 200 is a

directory of the other files stored in the system controller memory 69. The remaining memory is divided into a system status file 201, system data table 202 and a set of system support files 203.

The system status file 201 contains data relating to the configuration of the entire programmable controller 10. Included in this file is information identifying the various user selectable features of the programmable controller that have been enabled by the system operator. The real time clock data regarding the time of day, month, day and year are also included in this portion of the system memory. Digital words indicating the occurrence and type of various system faults and errors, as well as pointers indicating the program instruction being executed when the fault occurs are stored within another sub-file of this section. A section of the system status file 201 also lists the number and type of all the active modules on the system as well as the relative module number and address pointers necessary to access each module. For example, if more than one program processor module 18 or remote I/O scanner module 20 is present in rack 12, the user must assign a unique number via a thumb wheel on the module to distinguish the various modules of that type. The thumb wheel setting is read by the system controller during initial start-up of the system and stored in this portion of the system status file 201.

The system data table 202 contains data that is shared by more than one module. For example, results of various computations from one program processor module 18 may be stored in this portion of the system memory so that other program processor modules may readily access the data. Memory space is allocated within the system data table 202 to store the data that was received or that will be transmitted via the various external communication links of the system controller's communication section. Other modules in the system 10 are directly able to access these storage locations.

The system data table 202 also contains the value of various system counters and variables which are either used by the system controller 16 or which are commonly used by a number of other modules such as program processor modules 18 or the I/O scanner modules 20. The final sub-file within the system data table 201 is a space allocated for the user defined data for various programs that the user has loaded into the programmable controller.

The final section 203 of the system controller main memory 69 is dedicated to the system support files. These files include the source program information for the function chart program. The programmable controller 10 does not directly execute the function chart program. However, as will be described later, the function chart is employed during the programming step to generate data which is used to direct the operation of the program execution modules 18. In order to permit the subsequent editing of these programs, a source version of the function chart must be available for display on the programming terminal. As also will be described hereinafter, the support files 203 contain simultaneous counters for execution of various branches of the function chart. Although the local memory in each module contains data regarding its status, in some instances these memories do not have a battery to sustain them. In order to retain such volatile information after a power shut-down, the status information for these modules is replicated in a sub-file of section 203 of the system memory 69.

Communication parameters are also stored in this section 203 for configuring the UART 46 and the serial I/O module 48 within the communication section of the system controller 16. Among other things these parameters include baud rate, word size and control bits for the serial data signal format. For example, parameters for communicating with the operator terminal 24 are stored in this portion of the system memory. In addition, as noted previously, a number of programmable controllers may be connected via the local area network 28, in which case, parameters must be provided in each controller instructing them how to communicate over the network.

Referring again to FIG. 3, the processor section of the system controller 16 interfaces with the backplane buses of rack 12 via a plurality of components that are coupled to both sets of buses. Specifically, the backplane data bus 22 is connected to the processor data bus 62 by a set of bidirectional data transmission gates 78 and the backplane address bus 23 is connected to the processor address bus 63 by another set of bidirectional gates 76. When the system controller 16 seeks to exercise control over the backplane 11 of the programmable controller 10, a master mode control circuit 81 responds to signals on the control lines of the processor bus 61 and issues the proper control signals over the backplane control bus 21 to access other modules within the rack 12.

When another module within the rack 12 seeks access to the system controller 16, in order to read the contents of main RAM 69, for example, the system controller becomes subordinate to the control of the backplane 11 by this other module. In this circumstance, a slave mode control circuit 82 within the system controller 16 responds to the address of the system controller that appears on the backplane address bus 23 and to control signals on the control lines of the backplane bus 21 which lead from the other module. In response the slave mode control 82 issues signals to transmission gates 76 and 78 enabling the other backplane module to access the system controller 16. In this latter instance, the master mode control circuit 81 is in a dormant state. The two bus gates 76 and 78 receive enabling control signals from the master or slave mode control circuits 81 and 82 via the lines of control bus 61 depending upon the mode of backplane communication. A backplane arbitration circuit 84 supervises access to the backplane and resolves conflicting requests for access from the modules in the system.

As noted above the system controller module 16 executes programs which control the initialization of the system and communication with external computers. It does not execute the user control programs.

Program Execution Processor

The program execution processor modules 18 store and execute specific user control programs, such as ladder programs. One of these modules is shown schematically in FIG. 4. During this execution the modules 18 read the state of the sensing devices from input image table in the memory 134 of the various I/O scanner modules 20, and write output data from its memory to the output image tables in the I/O scanner modules. Data is also obtained from the system memory in the system controller 16.

In order to perform these tasks, each processor module 18 has a set of internal buses 91–93 which are coupled to the backplane 11. Specifically the processor

module 18 has a thirty-two bit internal data bus 92, a set of control lines 91 and a sixteen bit address bus 93. These are coupled to the data and address buses of the backplane 11 by respective set of tri-state, bidirectional transmission gates 94 and 96. The operation of these gates 94 and 96 is governed by an interbus control circuit 95 coupled to backplane control lines 21 and the module control lines 91. It should be noted that both the internal data bus 92 and the backplane data bus 22 are thirty-two bits wide. Therefore, thirty-two bit data from the processor module 18 can be sent over the backplane as one thirty-two bit word if the recipient module has a thirty-two bit wide data bus. In some processing functions the module 18 operates on sixteen bit data, and in such case sixteen-bit words are applied to the backplane 11.

The remaining components of the processor module 18 are connected to the internal buses 91–93. These internal buses are driven by a microprocessor 98, which is a thirty-two bit microprocessor sold commercially by Motorola, Inc. as the 68020 microprocessor. The microprocessor 98 has an interrupt port which is coupled to an interrupt interface circuit 99. This interface circuit receives signals from four external interrupt lines connected to terminals on the front of the processor module 18. These external interrupt lines permit devices which sense high priority events, to be interfaced directly to the processor module for fast response. Three other interrupt lines on the interface circuit 99 connect to circuits within the processor module 18. A signal on any of these external or internal interrupt lines causes the microprocessor 98 to immediately interrupt normal program execution and execute a routine that corresponds to that interrupt line. The user may program the routines for the external interrupts, but the internal interrupts are serviced by dedicated interrupt service routines.

The processing capability of the processor module 18 is also supported by a floating point co-processor 100, and by a bit co-processor 102. The floating point co-processor 100 is commercially available from Motorola, Inc. as the 6881, and it is specifically designed to work with the 68020 microprocessor 98. The bit co-processor 102 is a custom integrated circuit for carrying out Boolean logic operations on individual bits of the data words. Bit co-processors have been used in programmable controllers in the past to execute a set of ladder diagram instructions using hardwired logic as described in co-pending U.S. patent application Ser. No. 717,221 filed on Mar. 28, 1985 and entitled "Programmable Controller with Function Chart Interpreter".

The three processors 98, 100 and 102 operate in tandem to execute specific types of instructions included in the control program. The microprocessor 98 may begin execution of the control program and when it encounters a floating point arithmetic function, the floating point co-processor 100 is enabled and the CPU 98 relinquishes the internal buses 91–93 to it. The floating point co-processor 100 takes over the processing function until the arithmetic operation is complete at which time the microprocessor 98 resumes program execution. If the control program calls for bit processing (i.e. contains an instruction in the set for the bit co-processor 102), the microprocessor immediately relinquishes control to the bit co-processor 102 by writing the address of the control program instruction into a program counter in the bit co-processor 102. The bit coprocessor 102 then removes the microprocessor 98 from the internal buses 91–93 and executes the subsequent control program instructions until a stop instruction is encountered. At this point the bit co-processor 102 signals the microprocessor 98 via the control bus 91 to resume control of the buses and execution of the control program. Approximately 85–90 percent of a typical control program of the "ladder" type may be executed by the bit co-processor 102. The operation of the custom Boolean logic bit co-processor 102 in conjunction with a microprocessor is fully described in above-cited copending U.S. patent application Ser. No. 717,221.

The processor module 18 includes a data size acknowledge (DASACK) circuit 108. This circuit provides a two-bit code on two of the control bus lines which indicates the "width" of the data on the data bus 92. As will be described in more detail below, this data may be a long word consisting of thirty-two bits, a regular sixteen bit word, or a single eight bit byte. This data size information is used by the various module components in this data processing.

The final component of the processor module 18 is a control and status circuit 110 which monitors the status of the processor module and provides proper control signals on various lines of the control bus 91 to enable various components in a conventional manner.

Both a read only memory (ROM) 104 and a read-write random access memory (RAM) 106 are connected to all three internal buses 91–93 within the processor module 18. The ROM 104 contains 16 bit storage locations for instructions and constants for the three processors 98, 100, and 102. The RAM 106 provides storage for the operands and the results of the various computations performed by the processor module 18. The control programs to be executed by the module 18 are also contained in its RAM 106.

The details of the RAM memory 106 are shown in FIG. 5. The random access memory 106 is divided into lower and upper banks 112 and 114. Each bank contains a number of storage locations, for example 196,608 (192K) memory addresses. The memory location in each bank is sixteen bits wide and both banks can be enabled simultaneously to provide storage for thirty-two bit data words. As noted above the width of the data processed by the execution module 18 may be sixteen or thirty-two bits wide. In order to optimize the storage capacity when sixteen bit words are processed, a transmission gate multiplexer is incorporated into the random access memory 106 to allow separate sixteen bit words to be stored in both the upper and lower memory banks. Specifically, the multiplexer consists of three sets of tristate bidirectional transmission gates 116–118, each of which provides bidirectional control of sixteen data lines. The first set of transmission gates 116 couples the sixteen least significant bits (bits 0–15) of the data bus 92 to the data terminals of the lower memory bank 112. Similarly, the third set of transmission gates 118 couples the sixteen most significant bits (bits 16–31) of the data bus 92 to the data terminals of the upper memory bank 114. The second set of transmission gates 117 cross couples the sixteen least significant bits of the data bus to the data terminals of the upper memory bank 114. The three sets of transmission gates receive control signals from the DASACK 108 via control bus 91 which enables the various transmission gate sets depending upon the width and address of the word being sent on data bus 92. All of the address bus lines 93 go to each memory bank 112 and 114.

If thirty-two bits of data are being sent on the data bus 92, the DASACK 108 enables the first and third sets of transmission gates 116 and 118. This stores the sixteen least significant bits in the lower memory bank 112 and the sixteen most significant bits of the data in the upper memory bank 114. When a sixteen bit word is being sent on the data bus 92 it may be stored in either of the memory banks 112 or 114. If it is to be stored in the lower memory bank 112, the DASACK 108 enables only the first set of transmission gates 116 to pass the data to that memory bank. To maximize the storage capability for sixteen bit words, the separate data may also be stored at the same address in the upper memory bank 114, in which case DASACK 108 enables only the second set of transmission gates 117 which couples the sixteen least significant bits of data to the upper memory bank. When sixteen bits of data are being processed, the third set of transmission gates 118 is never enabled.

FIG. 9 represents the data structure of the RAM memory 106 for each program execution processor module 18. The memory 106 includes a section 310 which contains status information regarding the module's operation. Each program execution processor module 18 also contains its own data table 312 which is stored in the RAM 106. The data table 312 includes memory locations for various counters, timers and intermediate computation values.

The largest share of the RAM memory 106 is devoted to storing the control programs. The actual program contents, as will be described in detail later, comprise compiled user control programs, independent background tasks and various interrupt routines to be processed by the processor modules 18. In order to properly carry out the user control programs, support files containing the function chart data, called "descriptors," are also contained within the program area 313.

In one mode of operation of the program execution processor module 18, referred to herein as the "synchronous mode", the processor module 18 periodically copies the entire input image table from the I/O scanner module 20 into its own memory 106. Space for this copy of the I/O image table is provided in memory section 314.

Remote I/O Scanner Module

As noted above, the I/O scanner modules gather input sensor data for use by the program execution processor modules 18. Referring to FIG. 1, 2 and 6, a remote I/O scanner module 20 couples the programmable controller 10 to one or more remote input/output racks 17 containing individual I/O modules 19 which interface the sensors, or input devices, and accuators, or output devices, to the programmable controller 10. Each scanner module 20 periodically requests input data pertaining to the status of the input devices connected to the remote I/O racks 17 and stores it in the module's input image table for reading by other controller modules, such as the processor modules 18. The scanner module 20 also contains an image table of output data that it receives from other controller modules, such as the processor modules 18. At regular intervals the updated output data in the scanner module's output image table is transferred to the respective remote input/output racks 17 to control the various actuators connected to these racks.

Referring specifically to FIG. 6, each remote I/O scanner module 20 connects to the three backplane buses 21–23. The I/O scanner 20 contains three sets of internal buses: memory access buses 121–123, microprocessor buses 131–133 and I/O buses 141–143. The three memory buses 121–123 are connected to the backplane 11 by a set of address bus gates 124 and a set of data bus gates 126. Both of these transmission gates are controlled by an inter-bus control circuit 128 which sends signals to the gates 124 and 126 via the memory control bus 121. A local random access memory, referred to as main RAM 134, is coupled to the three memory buses 121–123. It stores the input image table for the sensor information being input to the I/O scanner 20 from the remote I/O racks 17 and it also stores the output image table for the output data being output to the remote I/O racks 17.

FIG. 8 shows in detail the data structures stored in the main RAM 134 of each I/O scanner module 20. These data structures include an I/O image table for the remote sensors and actuators serviced by that module 20. The input image table 210 represents the sensor data and consists of three separate sections 212–214. The first section 212 is the image of the actual state of the various sensing devices. The information relating to the inputs that are forced on is contained in the second section 213 within the input image table 210. As with previous programmable controllers, the user may force the status of a given sensor to appear to be either on or off regardless of its actual state. This enables the bypassing of faulty sensors, for example. Forced on sensors are designated by a binary one in an address for each such input.

The sensors that are forced off are indicated in the third section 214 of the input image table 210 by a logical zero stored for those sensors. Although by definition the user may write into the forced data tables 213 and 214, the user is prohibited from writing into the actual input image table 212. During the operation of the programmable controller, the user programs can read either the actual input image data from section 212 or the forced image of the sensor. If the forced image is read, the scanner module 20 logically OR's the actual sensor input state with the forced on data from section 213, then that result is ANDed with the forced off data for that sensor from section 214.

The output image table 211, also stored in the main RAM 134, includes the output image data table 215 which represents the status for the output devices connected to the remote I/O racks 17 serviced by the I/O scanner module 20. Typically, this output data is determined by the execution of the user control program in the processor module 18. A second section 216 representing the forced on output data and a third section 217 representing the forced off output data are also included in the output table 211. This allows the user to define a given actuator as always being on or off regardless of the results from the execution of the user control program. For example, this is useful where a portion of the controlled equipment may have to be shut down for maintenance and should not be turned on by the user control program. The control program may read each of the output tables 215–217 individually. If the forcing of the output states is disabled the data sent to the remote I/O racks 17 for activating or deactivating the various controlled devices is obtained from the output image table 215. If output state forcing is enabled then the data sent to the remote I/O racks 17 is a logical combination of the three output tables 215–217 using Boolean logic that is identical to the combination of the three input tables 212–214 described above.

Referring still to FIG. 8, the data structure in the main RAM 134 of the I/O scanner module 20 also includes a block 218 that contains data regarding the status of the communication adapter in each of the remote I/O racks 17 serviced by the module 20. This data is used during the transfer of information over the I/O links 15 with those remote I/O racks.

Although the state of most of the sensor and operating devices may be represented by a single binary bit, certain devices, such as position sensors and analog devices, produce or require information that comprises a plurality of digital words. These data may be transmitted to or from the remote I/O rack 17 into the I/O scanner module 20 as a large block of data. Memory section 219 in the main RAM 134 contains information necessary to control such transfers of blocks of data and a companion section 220 provides a memory area for the storage of the actual blocks of data. For a detailed description of this block transfer reference is made to U.S. Pat. No. 4,413,319 entitled "Programmable Controller for Executing Block Transfer with Remote I/O Interface Racks".

Referring again to FIG. 6, the inter-bus control circuit 128 also sends control signals to an I/O data arbitration circuit 130 which resolves conflicting requests for access to the memory buses 121–123 from the backplane 11 and the microprocessor buses 131–133. Two sets of transmission gates, address gates 136 and bi-directional data gates 138, interconnect the memory buses 121–123 to the microprocessor buses 131–133 and receive control signals from the I/O data arbitration circuit 130 via the memory control bus 121. The operation of the remote I/O scanner 20 is controlled by an eight-bit microprocessor 140 which is connected to three microprocessor buses 131–133. Microprocessor 140 is commercially available from Zilog, Inc. as the Z80 and it is the only device which is directly connected to the microprocessor buses 131–133, other than the sets of transmission gates 136, 138, 144 and 146 and the I/O data arbitration circuit 130.

A set of address gates 144 interconnects the microprocessor address bus 133 to the I/O address bus 143 and a set of bi-directional gates 146 interconnect the data buses 132 and 142. Both sets of tri-state gates 144 and 146 receive control signals from the I/O data arbitration circuit 130 via the microprocessor control bus 131. The microprocessor control bus 131 is directly coupled to the I/O control bus 141.

The I/O control buses 141–143 interconnect the devices which interface the I/O scanner module 20 to the remote I/O racks 17. These include a serial I/O interface circuit 148 which provides two synchronous serial input/output channels 150 and 151, each of which has its own cable driver/receiver circuit 152 and 153 respectively. A counter/timer circuit (CTC) 154 and a direct memory access (DMA) controller 156 are also connected to the I/O buses 141–143. The I/O bus section of the scanner 20 also includes a random access memory, indicated as I/O RAM 158 for temporary storage of input and output data communicated over I/O serial links 15. A read-only memory (ROM) 160 is also connected to the I/O buses 141–143 and it contains the programs which are executed by the microprocessor 140 to carry out the functions of the scanner module 20. An address decode circuit 162 is also connected to the address and control buses in the I/O section of the scanner module 20 to interpret the addresses generated by the microprocessor 140 and produce the proper enabling and control signals on the lines of control buses 131 and 141.

The operation of the remote I/O scanner modules 20 is described in a subsequent section on data acquisition and transfer.

Controller Operation

The novel architecture of the present programmable controller 10 dictates that it functions in a unique manner. The areas in which the present system operates differently than previous programmable controllers include system initialization, I/O data acquisition and transfer, and intermodule communication. However, the most significant difference with respect to this controller lies in the formulation and execution of the user control programs. Each of these unique functions will be described in detail.

System Initialization

During system power-up, the system controller 16 supervises the configuration of the system and various other tasks as shown in the flowchart of FIG. 13. The first phase 450 of power-up occurs immediately after the system reset signal terminates. During this interval no module is allowed on the backplane 11 and the various modules perform local tests of their own memory and other subsystems. When each module is finished, it releases a common ready line of the backplane control bus 21. When the last module has completed its internal tests the signal on the ready line is true. This signals the system controller 16 to make a transition into the next power-up phase.

During the second phase 451, the system controller 16 goes on to the backplane 11 and requests the module in each slot of the rack 12 to identify itself and provide the system controller 16 with various information regarding the module, such as the module type, and addresses for various interrupts and data pointers which are stored in the system memory 69. Based on the information received from each module, the system controller 16 verifies that the system is properly configured. For example, if there is more than one program execution processor module 18 on the system, the system controller 16 verifies that each one has a unique module thumb wheel designation and that two of them do not have the same designation.

In the next phase 452, the system controller module 16 stores the backplane communication parameters in preassigned memory locations in each module. These parameters instruct the modules how to communicate over the backplane. Once a module knows how to communicate via the backplane 11 it loads its memory address pointers into memory locations in the system memory 69 of the system controller 16. These address pointers are used by other modules to access the data in the module or issue instruction to it. For example each module loads the addresses of its interrupts into the system status section 201 of the System Memory (FIG. 7). In the course of this phase, the various functional modules may communicate only with the system controller 16, not with any other type of module. Referring again to FIG. 13, the next phase 453 of the initialization allows the modules to talk with each other and exchange any necessary data. At the completion of this phase, the programmable controller 10 at step 454 makes a normal transition into the user defined start-up mode, typically either "program" or "run."

The system has three primary modes of operation: programming, program run and fault mode, as determined by the system controller module **16**. Each of these modes is subdivided into a number of internal modes. For example, the run mode has an initial input prescan internal mode which prior to program execution causes each of the remote I/O scanner modules **20** to gather data from their respective I/O racks **17** and create an initial input image table **210**. Following the input prescan, a program prescan is carried out during which one or more of the user control programs are scanned once to create an initial output image table **211**. Once both of these prescan modes have been completed and the input and output image tables **210** and **211** have been created, the various outputs are enabled and the run mode commences formal execution of the user control program.

In order to explain the formulation and execution of the user control program, one must understand how data and commands are transferred within the programmable controller **10**.

Data Acquisition and Transfer

Referring to FIGS. **1**, **2** and **6**, periodically each I/O module **20** gathers input data from sensors connected to each of the controller's remote racks **17**. This acquisition of data is similar to the I/O scans performed by previous programmable controllers such as the one described in U.S. Pat. No. 4,442,504. While the system is in the run mode, each I/O scanner module **20** sequentially requests each remote rack **17** to send input data regarding the status of sensing devices connected to the remote rack.

With reference to FIG. **6**, the gathering of data from the remote I/O racks **17** is carried out by microprocessor **140** instructing the I/O data arbitration circuit **130** to enable transmission gates **144** and **146**. This couples the microprocessor buses **131–133** to the I/O buses **141–143**. After the connection of the buses is completed, the microprocessor **140** sequentially sends commands to the remote I/O racks via SIO **148** and line drivers **152** and **153**. In response to these commands the remote racks **17** transmit their sensor input data to the I/O scanner module **20**. The received input data is temporarily stored in I/O RAM **158**. The communication protocol used to gather data from the remote I/O racks **17** is similar to that used by previous programmable controllers.

At regular intervals the gathered input data is transferred from the I/O RAM **158** to the input image table contained within the main RAM **134**. This transfer is accomplished by the microprocessor **140** requesting that the I/O data arbitration circuit **130** couple the memory access buses **121–123**, microprocessor buses **131–133** and I/O buses **141–143** together. If the backplane **11** is not coupled to the memory access buses **121–123**, the I/O data arbitration circuit **130** will issue control signals via control buses **121** and **131** to transmission gates **136**, **138**, **144** and **146**. In response to the control signals, the transmission gates interconnect the three sets of internal buses. The I/O data arbitration circuit **130** then sends an acknowledge signal to the microprocessor **140** indicating that the connection has been made. Upon receiving the acknowledge signal, the microprocessor **140** transfers the data between the I/O RAM **158** and the main RAM **134**. The input data is stored in the input image table **212** (FIG. **8**) of main RAM **134**. When the transfer is complete the microprocessor **140** signals the I/O data arbitration circuit **130** to disconnect the interconnection of the internal buses.

At other periodic intervals, the remote I/O scanner module **20** sends output state data to the remote I/O racks **17**. This process is similar to that described immediately above with respect to the input status data. The microprocessor **140** requests the interconnection of the internal module buses **121–123**, **131–133** and **141–143**. Once the interconnection is made the output image table data in main RAM **134** is transferred to the I/O RAM **158**. The bus interconnection is then disconnected. The output data in I/O RAM **158** is systematically sent to the remote I/O racks **17** over serial links **15** to activate or deactivate the operating devices on the controlled equipment.

As noted above, the I/O RAM **158** is used to store the data that is sent and received over the serial links **15**. During the communication process the microprocessor **140** does not have to access the main I/O image table stored in main RAM **134**. This freeing of the main RAM **134** from communication tasks as well as the I/O module's separate internal bus configuration permits other modules on the backplane **11**, such as the system controller **16** and processor modules **18**, to directly access the I/O image table data in the main RAM **134**. This direct access from the backplane does not involve the microprocessor **140** which may be simultaneously controlling the communication with the remote I/O racks **17**.

When the system controller **16** or one of the processor modules **18** desires to read the status of a given input device, it requests access to the backplane buses **21–23** in order to interrogate the I/O scanner module **20** that receives the input data from that particular sensor. Upon being granted access to the backplane **11**, the processor module **18** or system controller **16** addresses the appropriate I/O scanner module **20**. In response to this addressing, the interbus access controller **128** within the respective I/O scanner module **20** receives an access request signal over a line of the control bus **21** of the backplane **11**. The interbus access controller then signals the I/O data arbitration circuit **130** that a request to access main RAM **134** has been received from another module. At the appropriate time when the internal memory buses **121–123** are available to be connected to the backplane buses **21–23**, the I/O data arbitration circuit **130** signals the interbus access controller **128** that it may connect the backplane and memory buses together via transmission gates **124** and **126**. The completion of this connection is then acknowledged by the inter-bus control circuit **128** to the requesting module via a line of the backplane control bus **21**. The other module then reads from or writes data into the I/O data image table in the main RAM **134**. The data transmission may be a single data word or a block of words. The requesting module holds control of the backplane **11** until all of the requested data has been transmitted to it from the I/O scanner **20**. When the access is finished the inter-bus control circuit **128** is signaled via the backplane control bus **21** and the connection to the backplane **11** is disconnected.

As is seen, the configuration of the I/O scanner module **20** permits other modules connected to the backplane to have direct access to the I/O scanner's main RAM **134** without the intervention of the scanner's microprocessor **140**. This allows the microprocessor to devote its attention to controlling the gathering of sen-

sor data and transmitting output commands to the equipment actuators.

The access to the I/O image table data by each program execution processor **18** may be on an "as needed" basis or periodically an image of the entire I/O data table may be read by the processors from the scanner modules **20**. In the "as needed" mode whenever the user controller program being executed by the processor module **18** requires the evaluation of a sensor status, the processor module **18** gains access to the backplane buses **21–23** and requests data from that sensor via the appropriate I/O scanner **20**. This mode is referred to as "asynchronous" in that the access to the I/O scanner module **20** is on an "as needed" basis and not a periodic basis which is synchronized to the execution of the user program.

An alternative method of accessing the sensor input data from the I/O scanner modules **20** is in a synchronous, or periodic, fashion. At a given point within the execution of the user control program by the program execution processor **18**, the input image table **210** from each I/O scanner **20** is read and copied into the local memory **106** within the processor module **18**. For example, just prior to the commencement of each pass through the user control program, the processor module **18** gains access to the backplane **11** and interrogates each of the I/O scanner modules **20** copying each scanner's input image table **210** into the processor module memory **106**. This ensures that the input data being used during the subsequent pass through the user program will remain constant and that each rung will be evaluated using the same sensor status. The choice of which mode of operation to use is left to the operator/programmer and depends upon the characteristics of the particular function chart and user control programs being executed. Unless otherwise specified by the operator, the system defaults into the asynchronous mode.

Referring particularly to FIGS. **2** and **8**, the output image table **211** in each I/O scanner module **20** is updated immediately upon a change of the status of an actuator connected to one of the remote I/O racks for that module. Specifically, when the user's program calls for a change in status of a controlled actuator, the program execution processor **18** gains access to the I/O scanner module **20**, as described above, and reads from the I/O scanner RAM **134** the output image table word that contains the bit or bits to be changed. Once the processor module **18** receives a copy of that output image table word, it changes the corresponding portion and writes the altered word back into the output image table **211** of the same scanner module **20**. During this reading and writing of the I/O scanner module's RAM **134**, the program execution processor module **18** retains control of the backplane buses **21–23** so that no other module may read or alter the output image table **211** of that scanner module **20** while the processor module **18** is performing its output function. This ensures that the output image table **211** is not changed or that another execution module uses stale data. It is, however, up to the user, through the control programs, to ensure that a given controlled device is only being activated or deactivated by one processor module **18** at a time.

Once a module has relinquished its control of the backplane **11** another module in the system may access the backplane. If more than one module seeks such access at the same time, the backplane arbitration circuit **84** in the system controller **16** resolves the conflict. The arbitration circuit **84** implements the rotating priority scheme described above whereby the highest priority level for backplane access passes from one module to the next in rack slot number order. Initially, the module in the leftmost rack slot has the highest priority and the priority level decreases with each slot to the right. At this time, if a conflict exists between the modules in the second, fourth and sixth slots, the module in the second slot gains access. When a module is granted access, the priority levels shift one module slot to the right. Now the second module has the highest priority and the module in the first slot has the lowest priority. Then, if the modules in the first, third and eighth slots simultaneously seek access to the backplane **11**, the third module gets access as it has the highest priority among the three. Then the eighth module has access and finally the first module is granted access the backplane **11**. The priority keeps shifting whenever any one of the modules accesses the backplane **11**. After the module in the last slot has had the highest priority level, the highest level rotates back to the module in the first, or leftmost, slot.

The system controller memory **69** may also be directly addressed by other modules on the backplane **11** so that they may gain access to system data. During the execution of the user control program, the main functions performed by the system controller **16** relate to the handling and execution of communications on the local area network **28** and the programming terminal **24**.

A module requiring data from the system controller module **16** or an I/O scanner module **20** is able to access the main RAM's in these modules directly via the backplane **11**. However, when a module desires to send a block of data or a command to another module in the rack **12**, a different technique for communication between the two modules must be employed.

Inter-module Communication

As indicated above, I/O data may be transferred directly from one module on the backplane **11** to another module. This direct transfer is possible when data is being read from or written to very specific memory data structures, such as an I/O image table, in the other module. However, in the present multiprocessor system, other forms of messages which do not reside in predefined data structures are often conveyed between the modules. A more general purpose inter-module communication protocol is required for such messages. This process is referred to herein as sending "mail".

Within each module that is capable of receiving mail is a set of mailboxes as illustrated in FIG. **14**. The set comprises sections for "priority" and "regular" mail. Each section has sixteen mailboxes, each consisting of a two word storage location. These sixteen mailboxes allow the exchange of messages with modules in the eight slots of the main programmable controller rack **12** and with eight modules in an auxiliary rack (not shown). Each of the mailboxes in each section corresponds to one of the slots in the two racks. During the configuration process, each module writes the address of the top of its mailbox set into the area of the system status file **201** (FIG. **7**) which contains data for that module. The address of each module's mailbox interrupt is also stored in the system status portion **201** of the main system memory **69**.

A "priority mail" message forms an urgent command which the recipient module is to immediately carry out. The command is either executed within the mail handling task which detects the receipt of the command, or it is passed to a destination task for processing. The

priority mail message consists of two sixteen bit words as shown in FIG. 12A. The first bit of the first word is a zero designating the message as priority mail. The next seven bit field (CMD) of the first word contains the coded form of the command to be executed which is selected from the following list:

Command Code	Command Description
1H	Mailbox Command Acknowledgment
2H	Begin Function Chart Step Execution
3H	Begin Interrupt Routine Execution
4H	Change System Mode
5H	Change Power Up Phase
6H	End Power Up
7H	Halt Active System Operation

The remaining bits of the first word are not used by every command. The second word contains a pointer to a location within the recipient module local memory containing data for executing the command. For example, a priority mail message is sent by one program execution processor module 18 to inform another such module to commence the execution of a specific user program for a given function chart step. In this instance the command (CMD) in the first word is "2H" telling the recipient module to begin a new function chart step. The second word is an offset pointer for the table containing the address in the recipient's memory that contains information about the step and its control program. This information is passed to the function chart interpreter program in the recipient module.

Referring to FIGS. 12A and 14, in order to send mail the source module 230 first formulates the two word message in its memory. The transfer of the message is controlled by a mail handling task in the module, see also the flowchart of FIG. 15. The priority mail transfer process is initiated by the source module 230 accessing the system controller main RAM 69 via the backplane 11 to obtain the addresses of the mail interrupt and the top of the set of mailboxes 233 in the recipient module 232 at step 460. These addresses are stored in the system status section 201 of the system controller's main RAM 69 (FIG. 7). The source module 230 knows the index from the top of the mailbox set 233 in order to access its mailbox slot in the recipient module. The mail handling task in the source module 230 then again seeks access to the backplane 11 at step 461. When it receives access to the backplane 11 at step 462, the source module 230 checks the first word of its priority mailbox in the recipient module memory 234 at step 463. If the first word of the mailbox is non-zero indicating a previous message is still in the mailbox, the source module waits a period of time and tries again. When the first mailbox word is found to contain all zeroes, the source module 230 writes the two word message containing the command and address offset pointer into its slot in the recipient mailbox list at step 464 and sends an interrupt word to the recipient module's mailbox interrupt address at step 465.

The mail handling task in the recipient module 232 in response to the interrupt reads the priority mail entry. If the entry is a command to start a new function chart step, an entry is made in the queue for the function chart interpreter program. An acknowledgment is then returned to the source module 230 via an entry in its mailbox set and an interrupt. At this time the recipient module loads the source module's mailbox slot with zeroes preparing it to receive another message.

"Regular mail" typically is used for transferring more than two words of data, although command messages described above also may be sent to these mailbox slots. For example, regular mail is used during system start-up to send configuration data to each module. It is also used during normal operations to send messages that do not require immediate response or action. Still referring to FIG. 14, when a source module 230 has several words of data to transfer, it assembles the data into a block 236 in its local memory 238. It then stores, in another block of memory 240, all the parameters necessary for the destination module to access the data in the source module's memory and acknowledge access to the source module 230. These parameters have a predefined format similar to that used in previous programmable controllers and include the address of the data message, its length, the destination module's task which will use the data, and other information necessary to transfer the data.

The two word message is then formulated in the source module's memory 238. The format of the message is shown in FIG. 12B. The first bit of the first word is a binary one designating this as a regular mail message. The next three bits are all zeroes indicating a data block type message as opposed to a command. The remaining twelve bits of the first word and all sixteen bits of the second word contain the beginning address of the transfer parameters within the source module.

The source module 230 then notifies its mail handling task to send the regular mail message. As with priority mail, this task accesses the backplane 11 and tests its regular mailbox slot in the recipient module's memory to make sure that it is empty. When the first word of the mailbox slot in the destination module is all zeroes, the two word regular message is written into the slot. Once the mailbox slot in the recipient module is loaded, the interrupt word is sent to the recipient module's mail interrupt addesss.

The recipient module 232 upon being interrupted scans its mailboxes for the message. Unlike priority mail, the recipient module 232 does not immediately process the regular mail, but places the mail entry in a queue for handling when time permits. The handling of regular mail is a lower priority task as compared to the execution of the user control program. When the recipient module 232 has time available, it accesses the source module's memory 238 via the backplane 11 and reads the information in the parameter block 240. The recipient module then uses these parameters to copy the message data block 236 into its local memory 234 via the data acquisition process described above. After copying the data the recipient module 232 sends an acknowledgment via regular mail to the source module 230 and zeroes out the regular mailbox slot. The data then may be processed by the recipient module 232.

With an understanding as to how data and commands are sent between modules, the execution of the different types of programs can be described.

Program Formulation and Execution

The present programmable controller may execute several general types of programs, such as a machine operation program, independent background programs, interrupt subroutines and fault subroutines. The machine operation program is developed by the user of the programmable controller to apply it to his particular

machine or process. For very complex processes, the machine operation program is defined by a sequential function chart showing each major step of the process. A separate user control program is developed to perform the functions of each step. These user control programs may comprise ladder diagram programs, as well as control programs written in higher level languages. Because the preferred embodiment executes compiled versions of the user control programs, assembly language and high level languages, such as BASIC, may be employed to generate the source code for the user control program. The user control programs are assigned for execution to the different processor modules. However, the present programmable controller provides the coordination of this parallel processing in accordance with the single machine operation program.

The independent background tasks are user programs that are subordinate in execution to control programs and may be used for lengthy non-time critical operations such as data acquisition from other computers and report generation. Interrupt routines allow high priority operations to be executed upon the occurrence of a given event, while fault routines permit a graceful recovery from a detected error condition in either the programmable controller 10 or the equipment being controlled.

Each processor module 18 has its own operating system which runs completely independent of the other modules 18 in the system and which is unaffected by the processing of those other modules. Routines on the same processor module 18 share the resources of that module with the operating system deciding how much processing time is allocated to each one of the simultaneously running routines.

In order to provide orderly execution of the various types of programs by a processor module, a priority system is established. The highest priority level consists of various interrupt routines that are so time critical that their execution may never be interrupted. The next level includes all other interrupts which are normally run to completion except when a top priority interrupt or fault occurs.

Interrupt routines comprise processor input interrupts and selectable timed interrupts. Processor input interrupts are started by an externally generated input and are used for high priority routines which are intended to be executed only upon the occurrence of a specific input condition. To accommodate the processor input interrupts, each processor module 18 has an interrupt interface 99 (FIG. 4) which receives and handles interrupt signals from external devices. The selectable timed interrupt routines are started at regular specified intervals by the system's real time clock. A priority is associated with each interrupt and if more than one interrupt is pending, the one with the higher priority will be executed first.

The execution of the various other program types is carried out by grouping the programs and allocating a portion of the processing time to each group. A given amount of the execution time may be designated by the operator during system configuration for the execution of non-time critical operations; such as background tasks, handling of regular inter-module messages, communicating with other computers and miscellaneous tasks. These low priority operations will be collectively referred to as "background tasks," although it is understood that background tasks are but one type of these operations. Typically, the operator may select 20, 30, 40 or 50 percent of the processing time of each processor module 18 for handling these tasks. The operating system will time-slice between the various operations of this type during the time allocated for them. When the allocated time expires the processing of these tasks is suspended until the occurrence of another interval for their operation. In addition, if the processor module is not currently executing a control program, the entire processing time will be devoted to the background tasks as necessary.

The remaining processing time is allotted to the execution of the machine operation program. As this task is usually time critical, the amount of time designated for background task processing generally is a function of how much time must remain for the machine operation program. By carefully selecting the amount of processor time devoted to the background tasks, the operator can cause an execution pass through the machine operation program to occur at regular intervals, which may be desireable in certain machine control applications. The machine operation program continues running until a timed interrupt signals the start of a background task interval.

The present programmable controller provides an improved system for executing the machine operation control programs, therefore, these programs will be described in detail. A function chart program defines the overall control process as a sequence of steps and provides the mechanism for coordinating the simultaneous execution of different parts of a single machine operation program in parallel by multiple program execution processor modules. As per convention each step is followed by a transition condition which specifies when the step is completed. A separate user control program, such as a ladder program, is written for each step and transition of the function chart.

The machine operation programs are written on the intelligent terminal 24 or on a personal computer or host computer connected via the local area network 28. These programming devices contain the necessary software so that the programmer can author the function chart and the user control programs. The terminal or programming computer also compiles each user control program into machine language instructions for direct execution by one of the processor modules 18. If a ladder program is used as the user defined control program, the technique for authoring it is the same as for conventional programmable controllers. The only difference is the compilation of the completed program.

The authoring of the function chart is different for the present programmable controller than for previous ones. The function chart is still graphically constructed on the screen of the programming terminal 24 or the programming computer. The software and techniques for doing this are similar to that practiced with conventional programmable controllers. However, unlike previous controllers, the function chart does not produce executable code but rather generates a set of descriptor files which may be thought of as directives that instruct the programmable controller which user control program to execute, when to do so and which processor module to use.

An example of a function chart is shown in FIG. 10. Each processing step of the function chart is designated by a rectangle such as box 403, and each transition from one step to another is designated by a horizontal line, such as line 402. The initial step of the function chart is defined by a double rectangular box as illustrated by

box 400. Although the initial step 400 in the FIG. 10 function chart is at the top of the chart, it may be almost anywhere in the chart, with the exception of within a simultaneous construct, as will become apparent. As with function charts for previous programmable controllers, step 400 includes the name of a file in memory (e.g. FILE 1) that contains the user control program to be executed for that step. Unlike previous function charts, the box 400 also contains an indication (P1, P2, etc.) of the processor module 18, that will execute the user control program for that particular function chart step. The present programmable controller 10 has two processor modules 18, which are designated P1 and P2, although additional processor modules can be inserted in the rack 12. The processor module 18 assigned to the initial step and the user control program file number are eventually stored in the system status file 201 of the system controller's memory (FIG. 7) so that the programmable controller knows where to begin executing the machine operation program.

The user control diagram program for step 400 for example is executed by the first processor module P1 which repeatedly executes the program until a programmed condition is met. That condition is represented by a transition (such as at 402) immediately below the box (400) in the function chart. Typically the transition 402 is defined by a single rung ladder program, which is executed on the same processor module, e.g. P1, as the associated step. The assignment of the processor module for the transitions is automatically made by the function chart editing program. If this rung is found to be true, the execution of the user control program for step 400 ceases and the program execution advances to the next function chart step 403. The portion of the function chart in FIG. 10 containing step 400, transition 402 and step 403 is typically referred to as a "sequence" construct in that each step sequentially follows the other.

Following step 403 are three separate program branches, only one of which will be selected for execution depending upon the corresponding transition condition. This choice of one of many branches is referred to as a "selection" construct. The first branch includes an initial transition 404 that is defined by the user control program contained in file 12 and processing step 407 defined by the user control program contained in file 3. Step 407 is followed by a termination transition condition 410 that is contained in file 15. Similarly, the middle branch contains an initial transition 405, a processing step 408 followed by a termination transition 411. The third and final branch of the selection construct consists of initial transition 406, main processing step 409 and a termination transition 412. Following transition 412 is a GOTO statement which when executed causes the program to jump to the point where the designated label appears. In this case the program jumps to the label "BRAD" before step 419 at the bottom of the function chart. All of the initial branch transition files 12, 13 and 14 are stored on and executed by the same processor module (P1) as the previous function chart step (403). It should be noted that since only one of the three branches of the selection construct is executed, they all may be processed by a single processor module 18, in this case the first processor module P1. Although, different branches could be designated for execution by different processor modules 18.

Upon the completion of the previous function chart step 403, which preceeds a selection construct, the conditions of the initial transitions 404–406 in each branch are sequentially examined. The first initial transition which is found to be true determines which of the branches will then be executed. For example, if the condition defined by the user control program in file 13, transition 405, becomes true first then only the middle branch having step 408 will be executed. The completion of the user control program for step 408 is indicated by the termination transition 411 contained in file 16. When that transition becomes true, the program transfers to step 413 contained in file 6 which is executed on the second processor P2.

Once step 413 is completed as indicated by transition 414 the function chart enters what is referred to herein as a "simultaneous" construct. A simultaneous construct comprises a plurality of function chart branches each containing at least one step. As the name implies the branches are executed simultaneously. In this case three processor steps 415–417 are to be executed in unison. The first step 415 comprises the control program stored in file 7 which is to be executed on the first processor module P1. The program for the second branch step 416 is contained in file 8 and is to be executed on the first processor module P1, while the third branch step 417 which is represented by the control program in file 9 is assigned to the second processor module P2. Because files 7 and 8 are both assigned for execution by the first processor module P1, the user control program contained in each of the files will be concatenated (i.e. strung together to run sequentially). This concatenation is similar to that practiced in present programmable controllers. However, whereas previous programmable controllers would have to concatenate all of the simultaneous construct files, including file 9, the present system has assigned file 9 for execution by the second processor module P2 and the remaining two files 7 and 8 for execution by the first processor P1. If the programmable controller 10 contained three separate processor modules 18, the function chart step for each branch could be assigned for execution by a separate one of the modules.

As an example, the simultaneous construct portion of the function diagram in FIG. 10 could control a manufacturing process where the user control program in file 7 controls the manufacturing process by reading various sensors and in response thereto activates or deactivates various pieces of production equipment. File 8 may consist of a control program that displays on the terminal 24 the status of the process being controlled by the program in File 7. In this case, the user control program in File 9 may monitor other sensors and activate alarms should any of these sensors indicate that given manufacturing errors have occurred.

The transition out of the simultaneous construct section is indicated by what is referred to herein as a "converge" construct. This construct contains a single transition step 418 in which the user control program in file 19 is executed on the second processor module P2 following each scan of the user control program in file 9. This converge construct transition is automatically assigned by the programming process to the same processor (e.g. P2) as the rightmost simultaneous branch. When the transition 418 is true, the execution of each branch of the simultaneous construct ceases at the end of their current program scan. As noted above with respect to the data structure of the system controller 16 in FIG. 7, the system support file 203 contains a memory area for the step counters of the simultaneous por-

tions of the function diagram. One of these counters is loaded upon entry into the simultaneous section with the number of simultaneous processing steps in the construct. After the transition condition 418 is satisfied, this counter is decremented as each step 415–417 completes its program scan and comes to a halt. When the counter reaches zero, all of the simultaneous steps are completed and the transition to the next steps 419 and 421 following the converge construct can occur.

In prior art systems, the three function chart steps 415–417 (FIG. 10), which were simultaneously executed, had to be concatenated for execution by a single processor and the execution of a given user control program occurred only once per pass through all three programs. Therefore, with respect to the above example if the control program in File 9 was an alarm function, it would only be scanned at the completion of the scan of the control programs in Files 7 and 8. Whereas in the present multiple processor system which utilizes parallel processing, the user control program for the alarm function is repeatedly processed by its own processor module P2 without the intervention of other user control programs being concatenated with it. This provides more frequent monitoring of time-critical conditions than is possible in a single processor system.

The graphical representation of the function chart per se is not executed by the programmable controller. It is used, however, by the programming terminal software to assemble a set of data files for each of the processor modules 18. Specifically, the function chart is reduced to a series of descriptor files that describe the operations of a portion of the function chart. Each descriptor contains data which identifies the user control program for a step in the function chart, data which identifies the termination transition, and data which identifies the descriptor (and its processor module) that is to execute the next section of the function chart. These descriptor files are stored in the processor module 18 designated in the function chart. The function chart interpreter software in each processor module 18 uses the respective descriptor files to control the execution of each user program.

By way of example with reference to the exemplary function chart of FIG. 10, a master descriptor file table is generated by the programming terminal 24 as shown in FIG. 11A. The descriptors fall into four distinct catagories which correspond to the function chart construct (i.e. sequence, selection, simultaneous, and converge) that generated the descriptor. Referring to FIGS. 10 and 11A the first descriptor 430 represents the sequence portion of the function chart, and its contents are displayed on the right side of FIG. 11A. The first word of the descriptor contains several bits, T, which indicate the type of the descriptor file, in this case a sequence type. The remaining portion of the first word identifies the number of the file that contains the user control program to be executed for the corresponding function chart step. In this example, function chart step 400 contains the user control program in file 1 for the first processor module P1. The first descriptor 430 also contains the number of the transition file which specifies the condition that is to occur before the execution of the user control program should terminate. File 11 is the transition file number in the first descriptor 430 as specified at point 402 in the function chart. Two bits in the transition file number field allow the operator to designate if the transition is to be forced true or false regardless of the actual state of the condition. This is particularly useful in program debugging. The descriptor 430 also identifies the next descriptor file to be used upon the completion of the current descriptor file and the processor module 18 in which it is stored. In this example the first descriptor 430 designates "Descriptor 2" as the next descriptor file which is assigned for execution by the first processor module P1.

The second descriptor file 432 on FIG. 11A is generated from the selection construct of the function chart in FIG. 10. The first word in the second descriptor 432 indicates that it is a selection type and that file number 2 contains the user control program for this step. The remainder of the descriptor 2 contains an array of transitions and their corresponding next decriptor file number and processor. For example, as shown in the function chart of FIG. 10, the first element of the array indicates transition file 12 and its next descriptor is descriptor 3; the second element of the array indicates transition file 13 and next descriptor 4; and the final entry in the array indicates transition file 14 followed by the next descriptor 5.

The descriptors for the three selection branches (descriptors 3, 4, and 5) are of the sequence type and have the same format as the first descriptor 430. The sixth descriptor is designated as the next descriptor file number in both descriptors 3 and 4. However, the next descriptor file number in the descriptor 5 for the third selection branch is the tenth descriptor for step 419 because of the function chart GOTO statement in the function chart.

The next type of descriptor is represented by the sixth descriptor 434 in FIG. 11A, which is produced from the simultaneous construct of the FIG. 10 function chart. The sixth descriptor 434 contains information regarding its type, the user control program file number and a single transition condition file to be executed. This descriptor 434 also contains an array of the next descriptor files which indicate the descriptors for each of the simultaneous construct branches, in this example branch steps 415–417. Each of these next descriptors is executed by its designated processor module 18 when the transition condition occurs.

The final type of descriptor file is a converge descriptor which controls the execution of each simultaneously executing branch, for example, steps 415–417 of the function chart in FIG. 10. A converge descriptor is generated for each branch. The branch steps 415, 416 and 417 are represented by descriptors 7, 8 and 9 in FIG. 11A. All of the converge descriptor files, as shown by descriptions 435 and 436, contain a word having several bits (T) designating its type and the number of the file containing the ladder program for the function chart step. Each converge descriptor file also contains a pointer to the simultaneous counter address in the system support file 203 within the memory of the system controller module 16. As will become apparent in the course of the description of the function chart interpreter software relating to the converge portion, the simultaneous counter is used to determine when all of the simultaneous branches have completed their execution. All of the converge branch descriptor files also contain the number of the next descriptor file to be executed and the processor module containing it. The converge descriptor file for the rightmost branch, in this case the ninth converge descriptor 436, also identifies the file containing the transition condition upon the occurrence of which the simultaneous execution terminates.

Once all of the individual descriptors are assembled by the programming terminal 24, they are sorted into groups according to the particular processor module P1 or P2 that has been assigned to interpret them. The various user control programs which are specified in the descriptors are similarly sorted by processor module. The descriptor data and user control program files are then transferred by the programming terminal 24 into the proper processor module 18 in the programmable controller 10. For example, the function chart descriptors 1–5, 7 and 10 and the user control programs in files 1–5, 7, 10–17 and 20 are assembled into a single data structure, as shown in FIG. 11B. This data structure is downloaded into the program memory area 313 of the first program execution processor module P1. The descriptors and user control program files for the remaining steps and transitions are assembled into another data structure as shown in FIG. 11C for the second processor module P2.

Once the sorted files have been downloaded into the respective processor modules 18, the programmable controller 10 is placed in the "run" mode. Each processor module 18 contains a function chart interpreter program which interprets the descriptors stored in its RAM 106 and executes the user control programs called for by the descriptors. When the execution of a machine operation step terminates, such as when the transition condition is satisfied, the interpreter program may commence interpretation of the next descriptor file or notify another processor module 18 that contains the next descriptor to begin interpreting it. As required by the descriptors, one or more processor modules can execute different portions of the machine operation program simultaneously.

FIGS. 16–19 illustrate flow charts of the program which enables each processor module 18 (FIG. 4) to interpret and process the various types of descriptors. The processing begins at step 590 at the top of FIG. 16 with the processor module's microprocessor 98 inspecting a list of active descriptor file numbers stored in its RAM 106. This active list contains a designation of each descriptor file that is currently being interpreted by the processor module 18. If there are no descriptors listed, the processor module enters a dormant state at step 609 where it remains until it receives a processing command. If the list contains an entry, the top descriptor is gotten off the active list at step 592 and the remaining entries, if any, are moved up in the list. The microprocessor 98 then evaluates the bits which indicate the type of the descriptor. Based upon the type of descriptor, the program at step 600 branches to one of four sections depending upon whether the descriptor is a select, simultaneous, sequence or converge type descriptor. As will be elaborated upon, there are several types of converge descriptors which are evaluated at branch step 607 before branching further to the specific routine for that converge type.

The remaining portion of FIG. 16 is a flow chart for the execution of a sequence type descriptor file which will be described with reference to the first descriptor file 430 in FIG. 11A. This branch routine commences with the processor module P1 at step 601 making one execution pass through the user control program specified by the step file number within the descriptor. At the completion of the pass the file containing the transition condition indicated in the descriptor file is executed at decision block 602. If the condition has not happened, the descriptor is returned to the bottom of the active list in the processor RAM 106 at step 606. The program then returns to step 590 to get the next descriptor from the active list. If only one descriptor is on the active list, the same user control program will be immediately executed again. This loop continues until the transition condition has occurred. At this point, step 603, the microprocessor 98 interprets the information within the first descriptor 430 relating to the next descriptor file to determine which program execution processor 18 contains the next descriptor to be interpreted. If the same processor module (P1) is to execute the next descriptor, that descriptor file is obtained from RAM 106 at program step 604 and placed in the active list at step 605. The program returns to step 590 for the microprocessor to check the active list and obtain the next descriptor for processing.

If the next descriptor file is contained in the memory of another processor module 18, a command is sent at step 608 to that processor module via priority mail as described above. The command instructs the other processor module to begin interpreting the next descriptor. Once the information regarding the next descriptor has been transmitted to and acknowledged by the other processor module 18, the previously active processor module 18 goes into a dormant state at point 609 unless other descriptors remain on the active list. In this dormant state a processor module 18 may execute background and other low priority tasks as will be described.

In the exemplary function chart of FIG. 10, the selection construct is processed next by the first processor module P1. As the second descriptor 432 is a selection type the program branches to the routine shown in FIG. 17. With reference also to the schematic of the processor module in FIG. 4, the microprocessor 98 at the first step 610 of the routine determines the number of elements in the array of transition condition files. An array pointer address in RAM 106 is set at step 611 to the first element in the transition condition array. The ladder program designated by the step file number in the descriptor 432 is then executed by the processor module P2 at point 612.

At the completion of one scan through the user control program, the transition condition contained within the file specified by the first element of the array is evaluated by the microprocessor 98 at step 613 to determine if the condition is met. If the transition has not occurred, the address contained in the array pointer is checked at decision block 614 to determine whether it is pointing to the address in RAM 106 containing the last element of the array. Since it is not the last element, the array pointer is incremented at step 615 and the program returns to decision block 613. The transition condition designated by the next array element is checked at decision block 613. Assuming that none of the transition conditions specified in the array has occurred, this loop continues until the last transition condition in the array has been tested. At this point the execution of decision block 614 indicates that the array pointer is at the address of the last array element and the program advances to step 621 where the descriptor is returned to the bottom of the active list. The processor module program execution then returns to step 590 (FIG. 16) to process the descriptor at the top of the active list.

The user control program for the select descriptor continues to be executed until such time as one of the transition conditions in the descriptor array has occurred. At which point in time, this transition condition is found to have been met at step 613 and the program

branches to step 616. The microprocessor 98 at step 616 reads the array field pointed to by the array pointer which specifies the file number and location of the next descriptor to be processed. At step 617 this next descriptor specification is evaluated. If the next descriptor is stored on the same processor module 18 as the current descriptor, the microprocessor 98 will read the descriptor file from its RAM 106 at process block 618. The program adds the descriptor to the active list at step 620 returns to step 590 in FIG. 16 to process the next descriptor file.

If the next descriptor is stored in the RAM of another processor module 18 for execution, the program will branch from decision block 617 to step 619 where the current processor module P1 sends a command message by priority mail to the other processor module 18 specifying that it is to "wake up" and begin processing the next descriptor. This message specifies the file number containing the next descriptor. Upon receiving acknowledgment of the command message, if the active list is empty, the current processor module P1 enters a dormant state at step 609 until it is notified by another module 18 that it is to resume processing another descriptor.

Referring to the exemplary function chart of FIG. 10, each of the selection branches is represented by a separate sequential type descriptor, the third, fourth and fifth descriptors in FIG. 11A. The transition from whichever one of the selection branches was chosen to the next step 413 is a standard sequential type transition. Specifically, if step 408 was selected for execution, the user control program contained in file 4 will be repeatedly executed until the transition condition 411 contained in file 16 occurs. At this point the program execution transfers to function chart step 413. This transfer will be accomplished by the first processor module P1 sending a priority mail message to the second processor module P2 indicating that it is to commence execution of the sixth descriptor 434 (FIG. 11A).

The second processor module P2 contains a copy of the function chart interpreter program in its local RAM 106. Upon receiving the command message from the first processor module P1, the second processor P2 adds the sixth descriptor 434 to the bottom of its active list. This action also "wakes up" the second processor module P2 if it was in the dormant state causing the module to enter its interpreter program at step 590 (FIG. 16). When the second processor module P2 begins execution of the sixth descriptor 434, its function chart interpreter program will determine at step 600 (FIG. 16) that this descriptor is a simultaneous construct type and the program will transfer to the routine indicated by the flow chart of FIG. 18. The first program step 630 in the interpretation of the simultaneous function chart descriptor is the determination of the number of elements in the array of next descriptors which indicates the number of simultaneous steps to be performed. The contents of array pointer address in RAM 106 is then set to the address of the first element at process block 631. The user control program contained in file 6, is then executed by the second processor module P2 at step 632. At the completion of one scan of the user control program, the occurrence of the transition condition 414 contained in file 18 is tested at decision block 633. If the condition has not occurred, the descriptor is returned to the bottom of the active descriptor list at step 638, so that the user control program will be executed again.

When the transition condition 414 has occurred, the processing transfers to commence the execution of the various function chart branches. The first element in the array of next descriptors, descriptor 434, is read from RAM 106 by the microprocessor 98 at block 634 and the processor module 18 for that next descriptor is then determined at block 635. If the module is the same as the current one, the next descriptor file is read from RAM 106 at point 636 and in step 637 is added to the processor module's active descriptor list with any other user control programs to be simultaneously executed on this processor module.

If one of the next descriptors is to be executed on another processor module (e.g. P1), the descriptor interpretor program branches from decision block 635 to step 639 where the current processor module P2 transmits a command via a priority mail message to that other module. The command instructs the other processor module to begin interpreting the descriptor file stored in it. The program then returns through node 640 to decision block 641 where the microprocessor 98 in the first processor module P1 determines whether the last array element has been processed. If additional elements remain, the contents of the array pointer address are incremented by the microprocessor 98 at step 642 and the next descriptor file number is read from the array at step 634.

This process of evaluating each of the next descriptors continues for each element in the array. Once the last element has been processed by the currently operative processor module P2 at block 641, the program returns to step 590 to check the active descriptor list for more work.

In the exemplary function chart depicted in FIG. 10, the three simultaneous branches, steps 415–417, terminate in a converge construct. The converge construct contains a single transition condition 418 upon the occurrence of which the execution of all the branches ceases. Each branch is represented by a separate descriptor (Descriptors 7–9) stored in the RAM 106 of the processor module P1 or P2 which is designated by the user to perform respective step of the function chart. Descriptors 7 and 8 are stored in processor P1 for execution and the ninth descriptor is assigned to the second processor P2. One of the descriptors, in this example the ninth one 436 (FIG. 11A) contains the transition condition and the next descriptor file number.

The operation of the second processor module P2 will be initially described before discussing the simultaneous operation of the first processor module P1. The ninth descriptor 436 is stored in the second processor module P2. The interpretation of the ninth descriptor 436 begins on at step 592 FIG. 16 with the processor module's microprocessor 98 reading the descriptor from the active list and evaluating its type. For a converge type descriptor the program advances to step 607 where a branching occurs depending upon the particular type of converge descriptor as determined by the microprocessor 98. In this case the converge descriptor contains the transition condition file number and the program advances to node D of the routine shown in FIG. 19. This converge descriptor interpreter routine begins at block 660 by the microprocessor 98 setting a pointer to the address of the simultaneous counter contained within the system support file 203 of the system controller's main RAM 69 (FIG. 7). The first time through the coverage routine for a given descriptor, the microprocessor 98 at step 661 loads the simultaneous

counter address with the number of simultaneous branches being executed, in this case three. The second processor module P2 then begins execution of the user control program from file 9 at process block 662. At the completion of one pass through the user control program, the microprocessor 98 executes the program in the transition condition file specified in the ninth descriptor file 436. If this transition condition has not occurred at step 663, the program replaces the descriptor on the active list at step 683 and then returns to step 590 (FIG. 16).

However, if the transition condition has occurred, the second processor module P2 at step 664 sets a flag in the system controller RAM 69 indicating the end of the simultaneous execution. While the second module P2 has access to the system controller RAM 69, it decrements the simultaneous counter at step 665 indicating that the processing of its simultaneous branch has been completed. Next at step 666, an evaluation of the counter is made by the second processor module P2 to determine if the counter has reached zero. If the count is not zero the program returns to step 590 (FIG. 16) to see if other descriptors remain to process. As none are left for the second processor module P2, it enters a dormant state in step 609.

If all of the simultaneous execution is completed (i.e. the count at step 666 is zero), the second processor module P2 at step 667 examines which processor module is to execute the next descriptor. If the second processor module P2 is to begin executing the next descriptor file, its microprocessor 98 at process block 668 reads the next descriptor and adds it to the active list at step 670. Then the program returns to step 590 at the beginning of the routine in FIG. 16 where it processes the new descriptor. If the next descriptor is to be handled by another processor module (e.g. P1), the routine branches to step 669. The second module P2 sends a priority mail message to that other module P1 informing it to begin executing the next descriptor. Then the second processor module P2 enters the dormant state at step 609.

As previously stated, there are several types of converge descriptors. The descriptors for the branch containing function chart steps 415 and 416 (FIG. 10) do not contain any information regarding the transition condition (see the eighth description 435 of FIG. 11A). These descriptors are stored and interpreted on the first processor module P1. When these descriptors are evaluated by the microprocessor 98 at step 607 of FIG. 16, the interpretor program in the respective processor module 18 branches to node E of the routine illustrated in FIG. 19. In this routine an address pointer is set at block 680 to the simultaneous counter address in the system controller main RAM 69. The user control program is then executed by the microprocessor 98 at step 681. After each user control program scan, the end flag address in the system controller main RAM 69 is checked at step 682 to determine whether or not it has been set, thereby indicating that the control program processing is to terminate. If the flag is not set, the descriptor is returned to that processor's active list at step 683. If the flag is set, the simultaneous counter is decremented twice at step 665 to indicate that the execution of files 7 and 8 by this processor module P1 is ceasing. The first module P1 then at step 666 checks the simultaneous counter and proceeds as previously described with regard to steps 666–670.

As noted above, in addition to the function chart and user control programs the user also may define "background tasks" for a specific processor module 18 to process. These programs are used to perform lengthy non-time critical tasks without an adverse delay in the operation of the function chart control program. User defined background tasks include report generation and certain subordinate control tasks. Report data regarding the performance of controlled equipment may be prepared for transmission via LAN 28 to a host computer. Such reports are not so urgent that the control of the equipment must be suspended while they are prepared and sent. The background control tasks also are used for lengthy calculations which are similar to those found in the main function chart program, but which are not required for real time control.

A background task program may be invoked from a user control program, an interrupt routine such as a selectable timed interrupt, or from another background program. A percentage of the processing time for each processor module 18 has been allocated to performing these background tasks. This percentage is set by the user so that the execution of background tasks do not significantly affect the execution of the machine operation program. Periodically the processing of the user control program is interrupted by the real time clock and the background tasks are performed for a given interval of time. If the end of the background task execution period occurs before the completion of the background program, the execution of the background task will be suspended and resumed during the next background task interval. When a user control program is not being run on the processor module 18, the background task may run almost continuously. In normal operation the background tasks will run intermittently to completion, however, their execution may be aborted by the user control program or upon the occurrence of an error condition.

Because the present programmable controller system has multiple processor modules 18, no single processor is constantly devoted to running the user control programs. This results in time being available for the processor modules to perform background tasks, without such tasks adversely affecting the control of the manufacturing equipment. This is yet another benefit of the present parallel processing concept as applied to programmable controllers.

The present programmable controller 10 also provides a two mode power loss recovery mechanism. This recovery mechanism is activated whenever power is lost during the execution of the machine operation program, such as due to an electric power outage. The operator may select during system configuration whether, when the power is restored after such a loss, the program restarts at the beginning (i.e. the initial function chart step) or resumes execution at the start of the user control program that was being executed when the power failed. The operator's selection of the recovery mode is stored in the system status file 201 in the system controller's main RAM 69.

With reference to FIG. 3, the system status circuit 88 detects the power beginning to fail and interrupts the processor section's microprocessor 66. This causes the microprocessor 66 to execute on interrupt subroutine which stores the state of each processor module's execution in a non-volatile memory. This state data includes the file numbers of the descriptors currently being executed and the descriptors on each processor

module's active descriptor list. Information regarding any background tasks being executed is also saved.

When power is restored, if the resume mode is selected, the system controller notifies each processor module 18 to begin execution with the descriptor file number that was stored when the power failed. As noted above, the major modules in the system have internal batteries to keep their respective memories alive when the power is shut off. Therefore, the programs and I/O tables remain stored in the modules' memories during the power outage.

What is claimed is:

1. A programmable controller for operating a machine to carry out a plurality of programmed functions, which comprises:

a backplane bus having leads for conducting data signals, address signals and control signals;

a plurality of processing means connected to said backplane bus, each processing means being operable to simultaneously execute a separate user control program that directs the programmable controller to operate the machine to perform a specific function;

a memory means which stores program execution sequence data and a plurality of user control programs, said memory means coupled to said plurality of processing means;

means responsive to the program execution sequence data for controlling the execution of the user control programs by said plurality of processing means;

an I/O interface means connected to said backplane bus for coupling the programmable controller to I/O devices, and

a system controller for supervising the access of said plurality of processing means and said I/O interface means to said backplane bus.

2. The programmable controller as recited in claim 1 wherein at least one of said processing means comprises means for interrupting the program execution by that processing means in response to an interrupt signal from an apparatus that is external to said programmable controller.

3. The programmable controller as recited in claim 2 wherein at least one of said processing means further comprises means for executing an interrupt program routine after the program execution is interrupted by said means for interrupting.

4. The programmable controller as recited in claim 1 wherein said program execution sequence data comprises a series of descriptors, each descriptor containing an identification of a user control program, an identification of a transition condition which indicates when the execution of the identified user control program is to terminate, and an identification of the next descriptor to be processed and which processing means will process the next descriptor.

5. The programmable controller as recited in claim 4 wherein each of said processing means includes means for interpreting the descriptors.

6. The programmable controller as recited in claim 4 wherein each of said processing means includes means for forcing the transition conditions to be either true or false regardless of the actual state of the condition.

7. The programmable controller for operating a machine to carry out a plurality of programmed functions, which comprises:

a backplane bus having leads for conducting data signals, address signals and control signals;

a plurality of processor means connected to said backplane bus, each processor means having a memory for storing user control programs for execution on that processor means and program execution sequence data comprising a plurality of descriptors each identifying a user control program, a transition condition which indicates when the execution of the user control program should terminate, and the next user control program to be executed and which processor means is to execute the next user control program, each of said processor means having means to transmit program execution commands to other processor means;

an I/O interface means coupled to said backplane bus for coupling I/O devices to the programmable controller; and

means for controlling access to said backplane bus by said plurality of processor means and said I/O interface means.

8. The programmable controller as recited in claim 7 wherein said program sequence data further comprises a designation of one of the user control programs to be executed initially at the start of program execution.

9. The programmable controller as recited in claim 7 further comprising means for saving program execution status data for each processor means upon an electrical power failure so that programs which were executing when the power failure occurred can resume executing upon restoration of electrical power.

* * * * *

Appendix B

Ref. 2.

US006304895B1

(12) **United States Patent**
Schneider et al.

(10) **Patent No.: US 6,304,895 B1**
(45) **Date of Patent: Oct. 16, 2001**

(54) **METHOD AND SYSTEM FOR INTELLIGENTLY CONTROLLING A REMOTELY LOCATED COMPUTER**

(75) Inventors: **Walter J. Schneider**, Brier; **Warren C. Jones**, Renton; **Mark D. Sasten**, Duvall, all of WA (US)

(73) Assignee: **Apex Inc.**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/359,376**

(22) Filed: **Jul. 23, 1999**

Related U.S. Application Data

(63) Continuation-in-part of application No. 08/916,685, filed on Aug. 22, 1997, now abandoned.

(51) **Int. Cl.**[7] **G06F 13/00**
(52) **U.S. Cl.** **709/203**; 709/217
(58) **Field of Search** 709/201, 203, 709/217, 218, 219; 345/77, 87, 89, 97, 100, 343, 520

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,634,830	1/1972	Baskin	710/131
3,774,158	11/1973	Clark	340/825.02
3,955,188	5/1976	Viswanathan	345/29
4,078,249	3/1978	Lelke et al.	358/1.17
4,081,797	3/1978	Olson	348/570
4,150,429	4/1979	Ying	710/131
4,243,984	1/1981	Ackley et al.	345/22
4,313,176	1/1982	Cecil	379/93.05
4,479,122	10/1984	Redman et al.	379/102.01
4,550,386	10/1985	Hirosawa et al.	345/505

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

93 03 716.3	11/1993	(DE) .
0 174 099	3/1986	(EP) .
87/00317	1/1987	(WO) .
94/19749	9/1994	(WO) .
95/01055	1/1995	(WO) .

OTHER PUBLICATIONS

PCT International Search Report for PCT/US96/13772, International Filing date Aug. 22, 1996.

Switchback. TM. User Guide Pamphlet. Copyright., Mar. 1995.

APEX Desktop Concentrator Product Brochure, APEX PC Solutions, Redmond, Washington, dated believed to be prior to Aug. 15, 1994.

APEX PC Solutions User's Guide, Manual P/N 053–0006–01, 8001/KVM, Redmond, Washington, Apr. 1993.

Apex PC Solutions Product Brochure, APEX PC Solutions, Redmond, Washington.

DeKerf, T. and Davis, Gary D., "The Keyboard/Video Switch White Paper: A Close Look at Modern Keyboard/ Video Switching", Tron International Inc. and the Work Center Corporation, 1995.

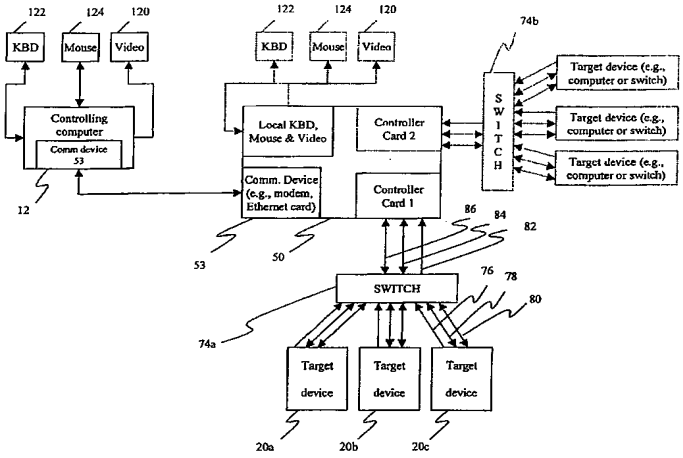
(List continued on next page.)

Primary Examiner—Moustafa M. Meky
(74) *Attorney, Agent, or Firm*—Oblon, Spivak, McClelland, Maier & Neustadt, P.C.

(57) **ABSTRACT**

A method and system for remotely accessing and controlling at least one of a target switch and a target computer using a target controller. The video information captured by the target controller is analyzed and compressed in order to reduce network traffic between the target controller and a controlling computer.

24 Claims, 13 Drawing Sheets



U.S. PATENT DOCUMENTS

4,599,611	7/1986	Bowker et al.	345/116
4,630,284	12/1986	Cooperman	375/257
4,641,262	2/1987	Bryan et al.	345/168
4,665,501	5/1987	Saldin et al.	710/8
4,680,622	7/1987	Barnes et al.	348/598
4,710,917	12/1987	Tompkins et al.	709/204
4,768,083	8/1988	Romesburg et al.	348/600
4,800,429	1/1989	Perkins	348/500
4,807,184	2/1989	Shelor	710/131
4,823,256	4/1989	Bishop et al.	714/10
4,879,716	11/1989	McNally et al.	710/131
4,907,079	3/1990	Turner et al.	725/11
4,939,507	7/1990	Beard et al.	345/156
4,941,087	7/1990	Kap	710/260
4,949,169	8/1990	Lumelsky et al.	348/14.12
4,949,248	8/1990	Caro	709/203
4,953,027	8/1990	Tong et al.	348/569
4,953,159	8/1990	Hayden et al.	370/265
5,008,747 *	4/1991	Carr et al.	375/240.12
5,029,111	7/1991	Mansell	345/550
5,036,484	7/1991	McCoy et al.	703/23
5,043,866	8/1991	Myre, Jr. et al.	707/202
5,051,720	9/1991	Kittirutsunetorn	340/310.02
5,117,225	5/1992	Wang	345/2
5,121,486	6/1992	Kurihara et al.	710/131
5,128,766	7/1992	Choi	348/564
5,166,674	11/1992	Baum et al.	714/752
5,214,785	5/1993	Fairweather	710/67
5,220,380	6/1993	Hirata et al.	399/8
5,222,212	6/1993	Johary et al.	345/520
5,230,066	7/1993	Morimi	345/501
5,247,364	9/1993	Banker et al.	348/569
5,247,615	9/1993	Mori et al.	709/205
5,260,778	11/1993	Kauffman et al.	725/33
5,261,079	11/1993	Celi, Jr.	703/24
5,268,676	12/1993	Asprey et al.	345/2
5,280,583	1/1994	Nakayama et al.	709/205
5,283,639	2/1994	Esch et al.	725/32
5,287,461	2/1994	Moore	709/219
5,301,028	4/1994	Banker et al.	348/570
5,317,391	5/1994	Banker et al.	725/139
5,327,156	7/1994	Masukane et al.	345/113
5,347,646	9/1994	Hirosawa et al.	714/49
5,349,675	9/1994	Fitzgerald et al.	379/102.01
5,357,276	10/1994	Banker et al.	725/102
5,357,420	10/1994	Gohi	700/2
5,367,571	11/1994	Bowen et al.	380/242
5,381,477	1/1995	Beyers, II et al.	380/231
5,392,400	2/1995	Berkowitz et al.	709/203
5,396,593	3/1995	Mori et al.	345/501
5,440,618	8/1995	Riegel et al.	379/93.04
5,440,632	8/1995	Bacon et al.	380/242
5,448,697	9/1995	Parks et al.	345/520
5,465,105	11/1995	Shatas et al.	345/204
5,477,262	12/1995	Banker et al.	725/38
5,485,221	1/1996	Banker et al.	348/563
5,486,868	1/1996	Shyu et al.	348/524
5,486,869	1/1996	Cooper	348/525
5,489,947	2/1996	Cooper	348/589
5,499,377	3/1996	Lee	709/244
5,502,499	3/1996	Birch et al.	348/523
5,504,522	4/1996	Setogawa	348/185
5,519,874	5/1996	Yamagishi et al.	709/201
5,526,024	6/1996	Gaglianello et al.	345/547
5,534,942	7/1996	Beyers et al.	348/569
5,537,548	7/1996	Fin et al.	709/204
5,539,822	7/1996	Lett	380/211
5,552,832 *	9/1996	Astle	375/240.24
5,566,339	10/1996	Perholtz et al.	713/340
5,577,210	11/1996	Abdous et al.	709/219
5,579,057	11/1996	Bander et al.	348/589
5,579,087	11/1996	Salgado	399/1
5,581,303	12/1996	Djabbari et al.	348/524
5,583,993	12/1996	Foster et al.	709/205
5,592,551	1/1997	Lett et al.	380/211
5,603,060	2/1997	Weinberger et al.	399/8
5,604,509	2/1997	Moore et al.	345/2
5,606,604	2/1997	Rosenblatt et al.	379/198
5,608,872	3/1997	Schwartz et al.	709/205
5,617,547	4/1997	Feeney et al.	710/131
5,642,153	6/1997	Chaney et al.	725/40
5,657,414	8/1997	Lett et al.	386/35
5,674,003	10/1997	Anderson et al.	709/228
5,689,671	11/1997	Stromberg	709/245
5,701,161	12/1997	Williams et al.	348/468
5,708,961	1/1998	Hylton et al.	725/81
5,715,515	2/1998	Akins et al.	725/142
5,719,622	2/1998	Conway	348/211
5,721,842	2/1998	Beasley et al.	710/131
5,724,525	3/1998	Beyers et al.	705/40
5,732,212	3/1998	Perholtz et al.	709/224
5,742,677	4/1998	Pinder et al.	380/242
5,757,424 *	5/1998	Frederick	348/218
5,768,224	6/1998	Tanaka et al.	369/33
5,774,859	6/1998	Houser et al.	704/275
5,801,789	9/1998	Zeidler et al.	348/589
5,884,096	3/1999	Beasley et al.	710/38
6,016,166 *	1/2000	Huang et al.	348/515

OTHER PUBLICATIONS

Switchback.TM. User Guide, Apex PC Solutions, 1995. Copyright.

"Sharp: Recording MD Player Due in Early '94", HFD the Weekly Home Furnishings Newspaper, Jul. 1994.

"Goldstar Unveils 5 Upgrade . . . ", HFD the Weekly Home Furnishings Newspaper, vol. 68, No. 10, Mar. 7, 1994.

"Electron Components: On Screen Display Ics", NEC Corporation, Apr. 1994.

"10comm ThinkSync II CM–2131T: Eye–Pleasing Trinitron CRT", Windows Sources: The Magazine for Windows Experst, vol. 1, No. 3, pp. 240–241 and 244, Apr. 1993.

"Genlocking" Tech Note, S3 Incorporated, Sep. 1993.

News Release "Maxi Switch, Inc. Introduces Industry's First Software Utility Permitting On–Screen Display of User–Prograied Keyboard Settings", Tucson, Arizona, Nov. 23, 1992.

PR Newswire, "Sony Makes Board Decision Statement with the Launch of the Trinitron XBR2 Line of Computer Televisions", New York, Oct. 15, 1992.

"Panasonic's New Video Line Offers More Styling, User–Friendly Features", HFD The Weekly Home Furnishings Newspaper, Jul. 1992.

Reachout: Remote Control for Windows and DOS, User Guide Version 2.1, Ocean Isle Software, Revised Jul. 2, 1992.

Lotus Brand Portable, Home TVs Bow, HFD The Weekly Home Furnishings Newspaper, vol. 64, No. 28, pp. 106 and 123, Jul. 9, 1990.

Gosch, J., "Solid–State Captions", Electronics, pp. 36–37, Apr. 1990.

Levine, J.A., "TV Makers Focus on Upscale Models", HFD The Weekly Home Furnishings Newspaper, Date unknown.

"MS Left Breathless by Jaunt through Eugene, OR, Video Countryside", Consumer Electronics, vol. 16, No. 4, Apr. 1988.

"Networking Software Master Net" Rose Electronics, 1988, COPYRGT.

CES '87 Consumer Electronics Show: New Products, Jan. 5, 1987.
"Panasonic VHS VCR Series Features Rounded Contours" HFD The Weekly Home Furnishings Newspaper, pp. 71 and 84, Jan. 5, 1987.
Master Link: Communication Utility for the PC, Rose Electronics, 1987.COPYRGT.
PR Newswire, "RCA Announces First Video Disc Player with Programmable Capability", Indianapolis, Aug. 12, 1983.
Buchsbaum, W., "RCA Model VGM 2023s 25" Color TV Receiver", Computers & Electronics, pp. 104, 106, 107, 109, Feb. 1983.
8001/KVM Users Guide, Apex PC Solutions, Date Unknown.
Addendum I to User's Manual for MasterNet Version 2.00, Rose Electronics, Revision A. Date Unknown.
Carrell, J.L. and Boyle, P.R., "ONline System Concentrator", PC Magazine. Date unknown.
MasterNet Networking Software Product Bulletin, "Zero Slot Lan Software Uses Sharing Device" and Instant Control of Your Peripherals Rose Electonics. Date Unknown.
Apex PC Solutions, "Apex/Desktop Concentrator" advertisement. Date unknown.
Apex PC Solutions advertisement. Date unknown.
Motorola Semiconductor Technical Data, "Advanced Monitor On–Screen Display CMOS" Rev. 2, Feb. 1997.
General Instrument 2750R Satellite Receiver User's Guide 2700 Series, Publication No. 72089–1, Rev. C, Apr. 1990.
Nguyen, Huy "Key–view unlocks server problems", PC Week Mar. 27, 1995 v12 n12 pN20.
Rigney, Steve "J&L's server room in a box", PC Magazine May 30, 1995 v14 n10 pNE19.

* cited by examiner

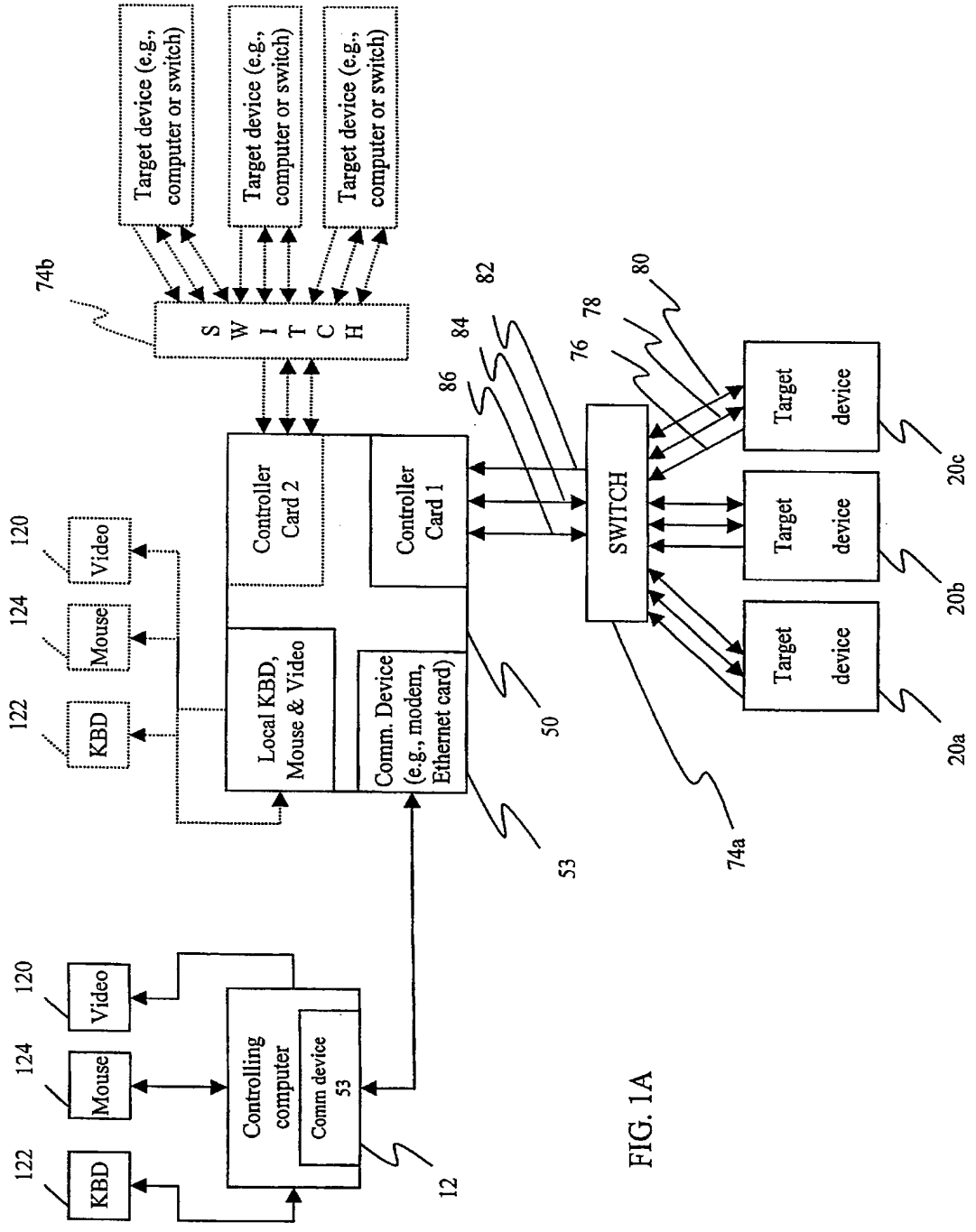

Target device (e.g., computer or switch)
Target device (e.g., computer or switch)
Target device (e.g., computer or switch)
74b
S W I T C H
82
84
86
76
78
80
Controller Card 2
Controller Card 1
SWITCH
Target device
Target device
Target device
20c
20b
20a
74a
120
Video
124
Mouse
122
KBD
Local KBD, Mouse & Video
Comm. Device (e.g., modem, Ethernet card)
50
53
Controlling computer
Comm device 53
12

FIG. 1A

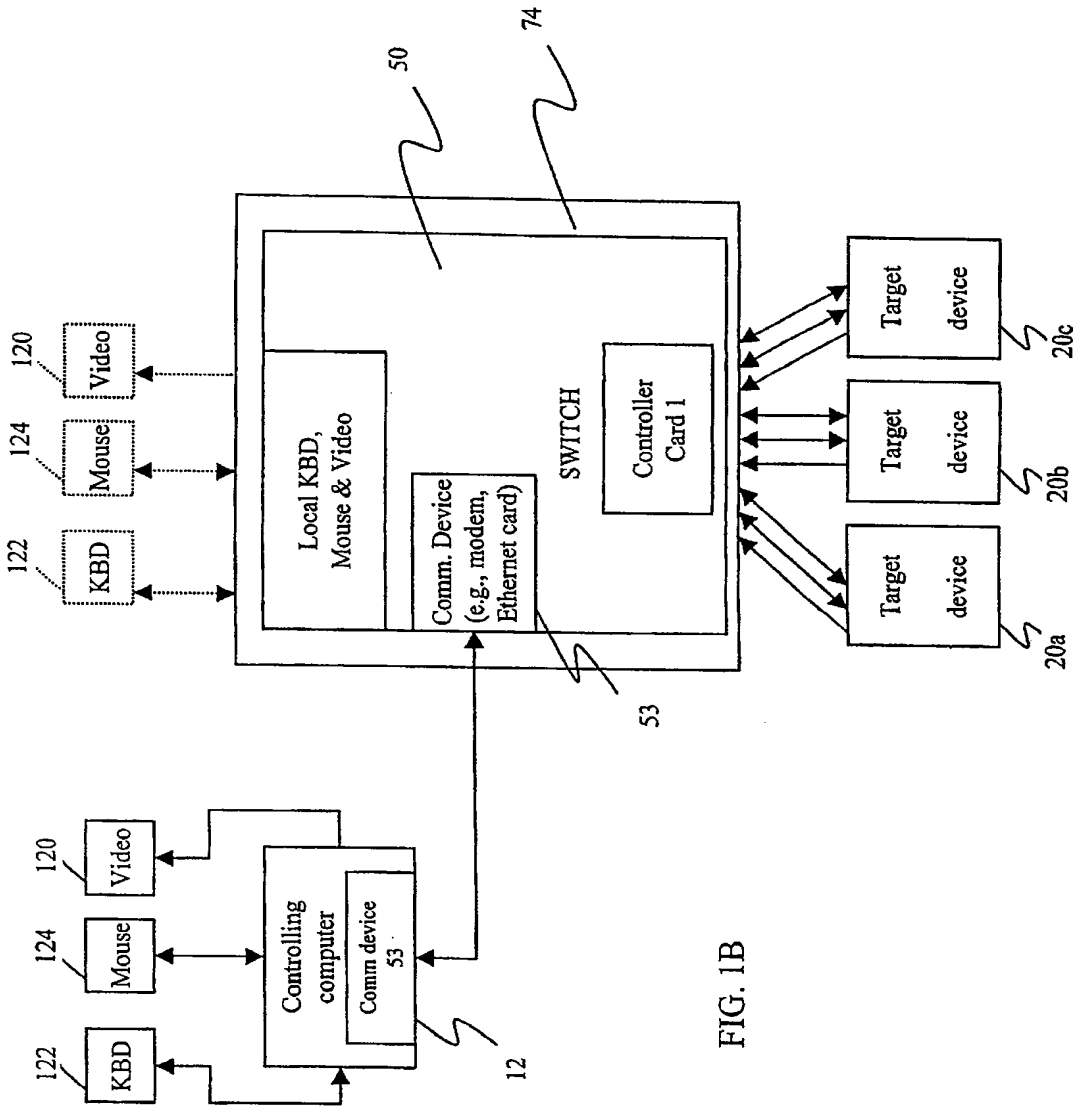

50
74
120
Video
124
Mouse
122
KBD
Local KBD, Mouse & Video
Comm. Device (e.g., modem, Ethernet card)
SWITCH
Controller Card 1
53
Target device
20c
Target device
20b
Target device
20a
120
Video
124
Mouse
122
KBD
Controlling computer
Comm device 53
12

FIG. 1B

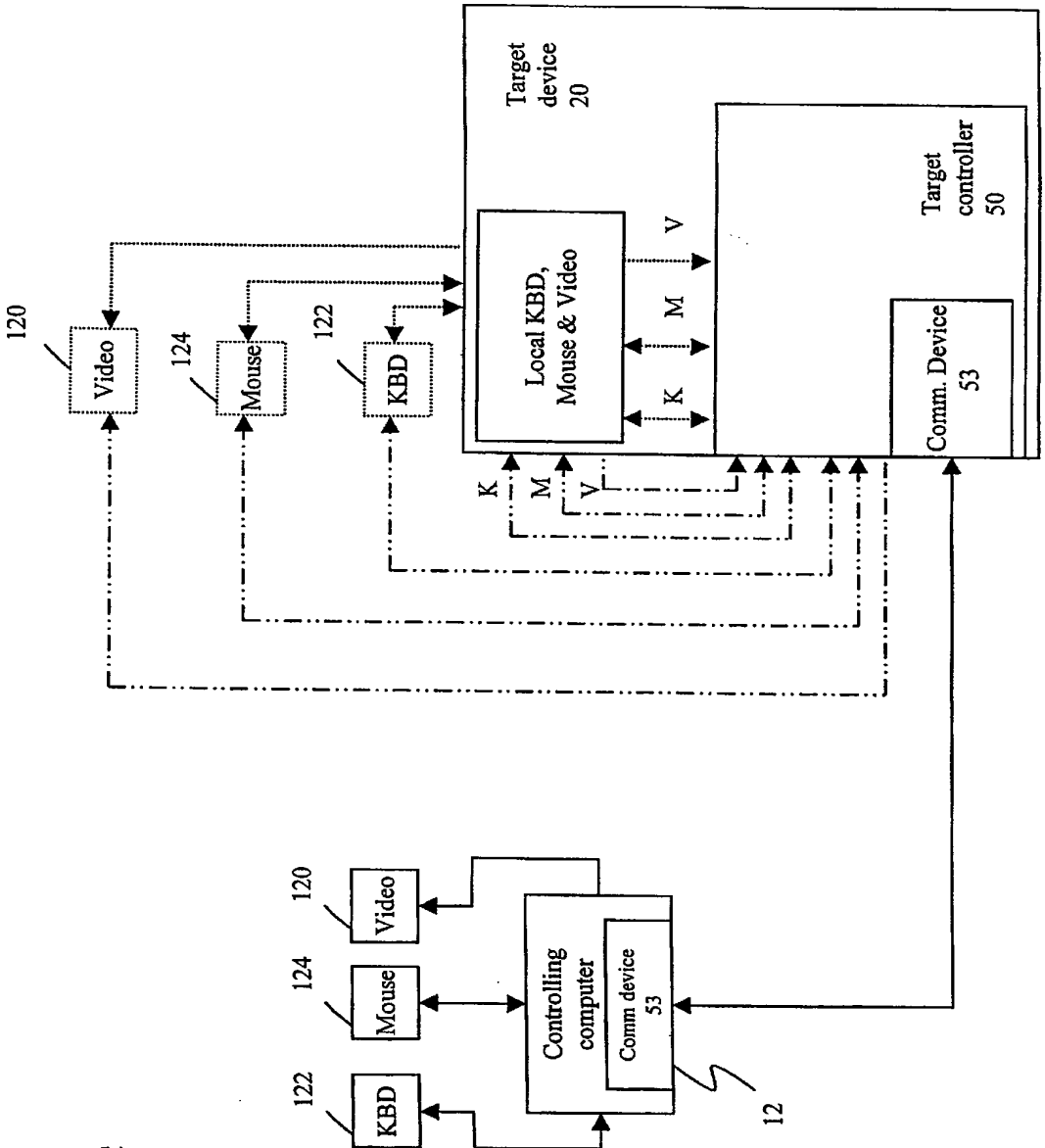

Target device 20
Target controller 50
Local KBD, Mouse & Video
Comm. Device 53
V
M
K
120
Video
124
Mouse
122
KBD
Controlling computer
Comm device 53
12

FIG. 1C

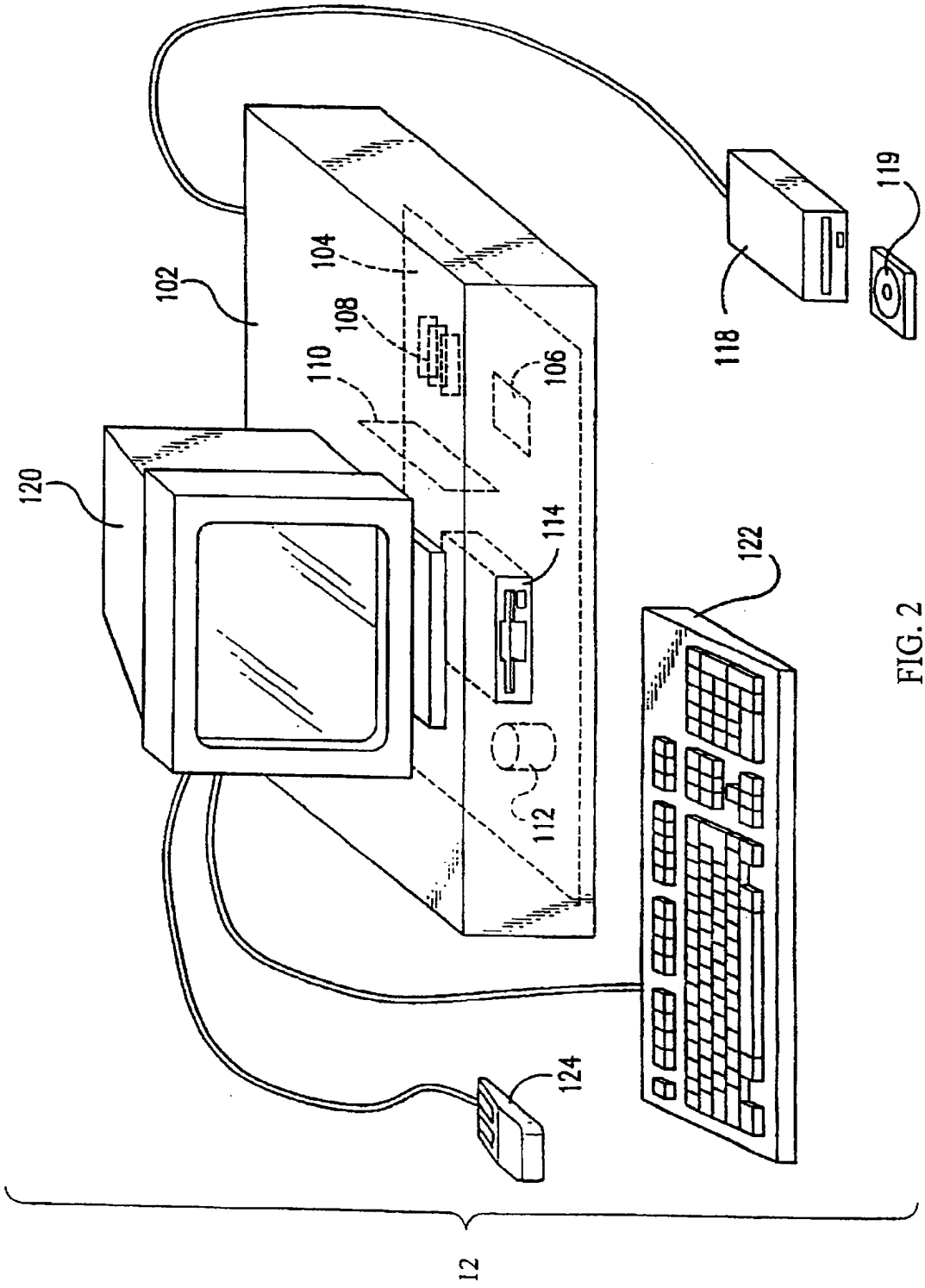


FIG. 2

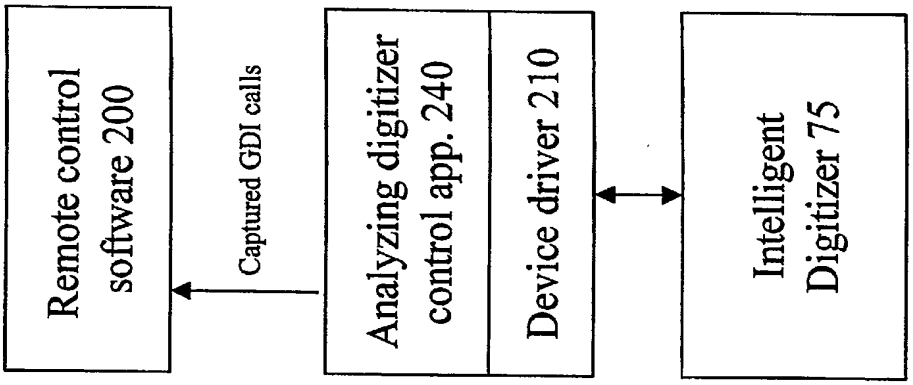


Fig. 3c

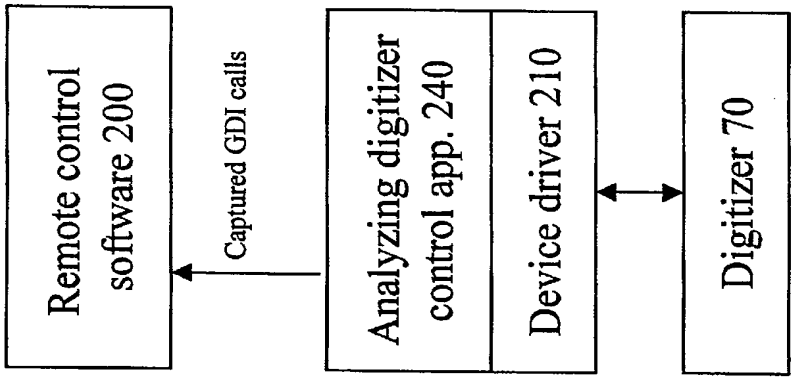


Fig. 3b

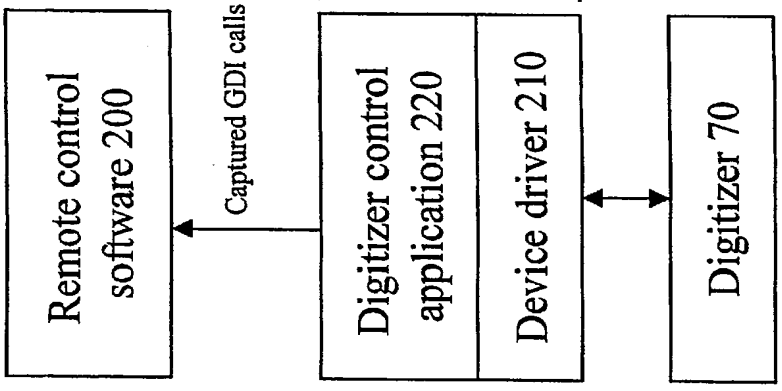


Fig. 3a

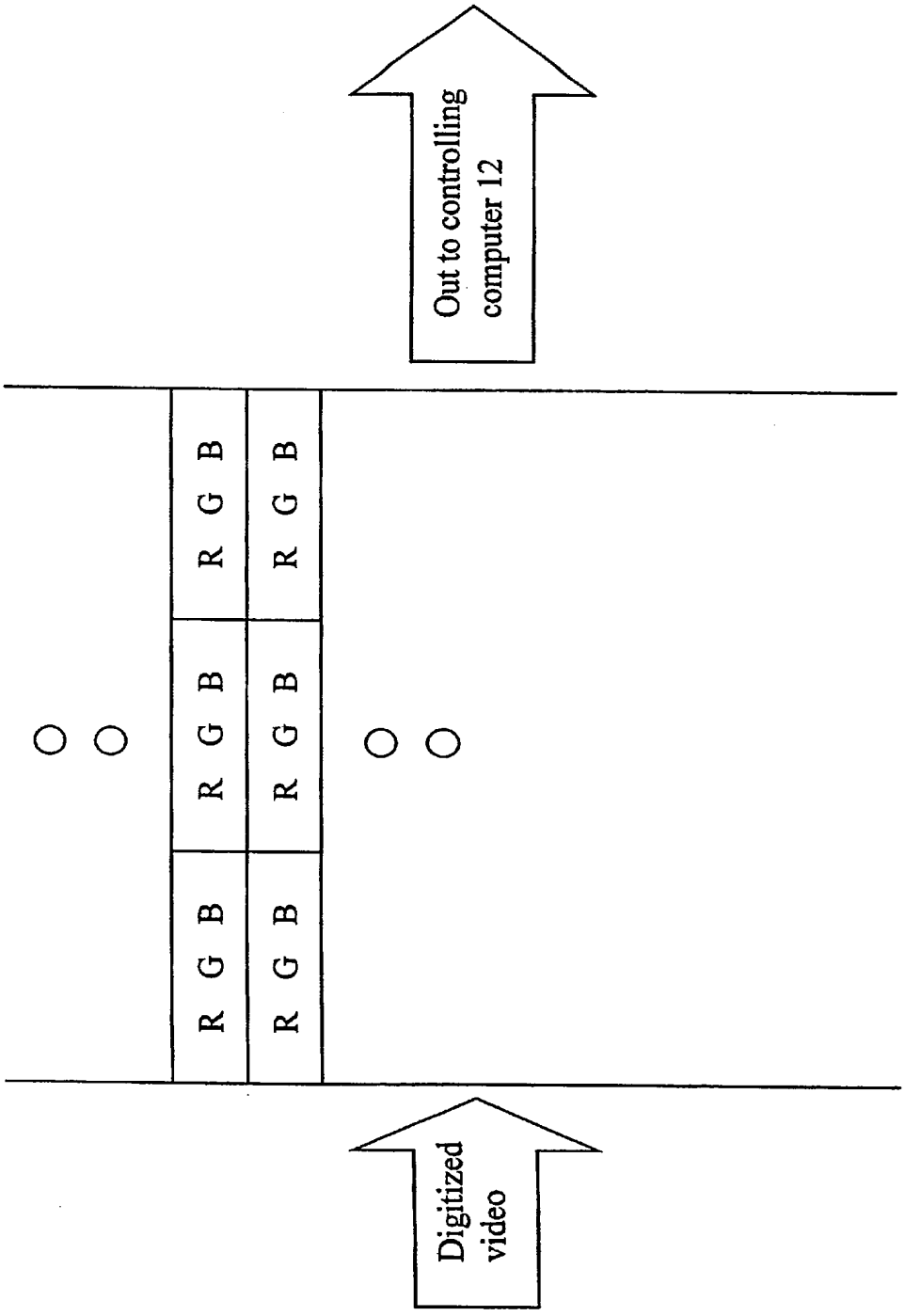

Out to controlling computer 12
R G B
R G B
R G B
R G B
R G B
R G B
Digitized video

Fig. 4

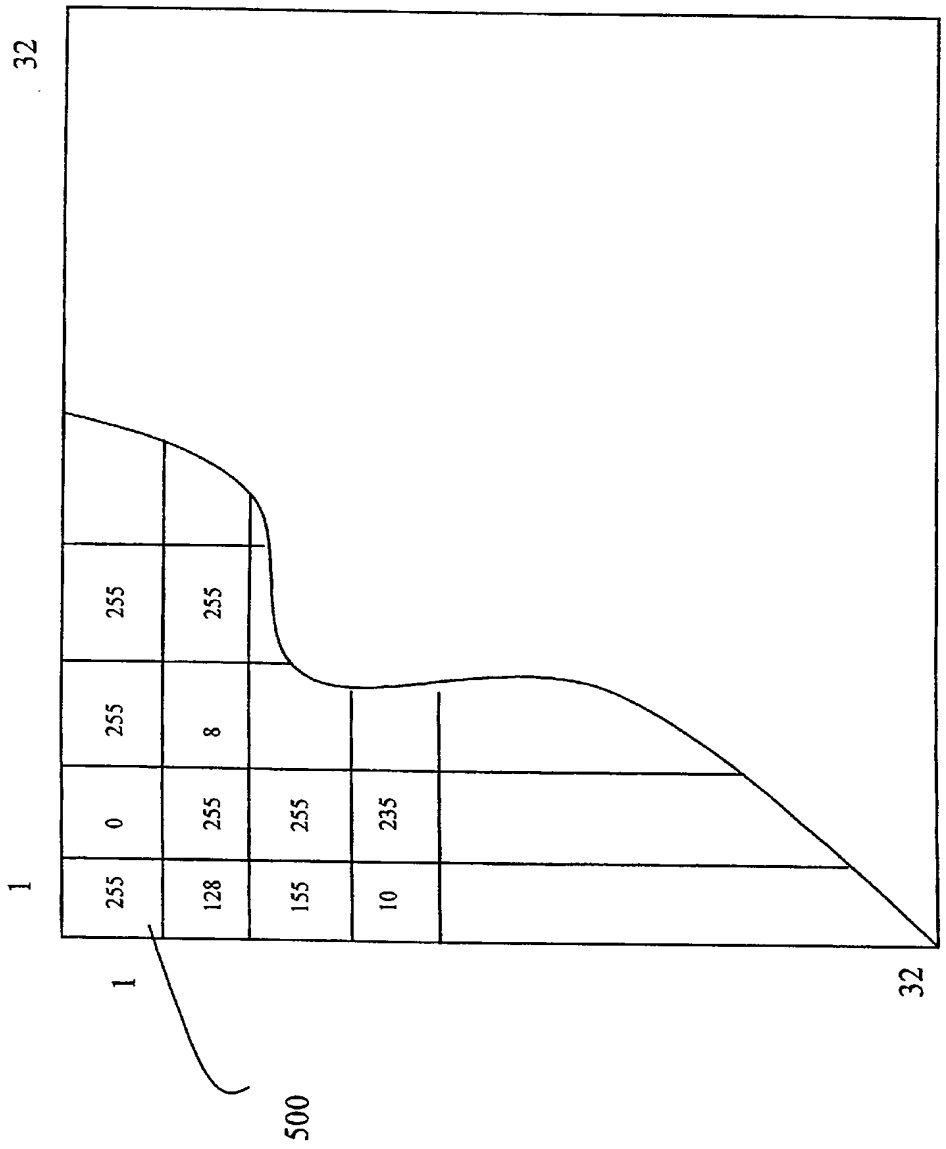


Fig. 5A

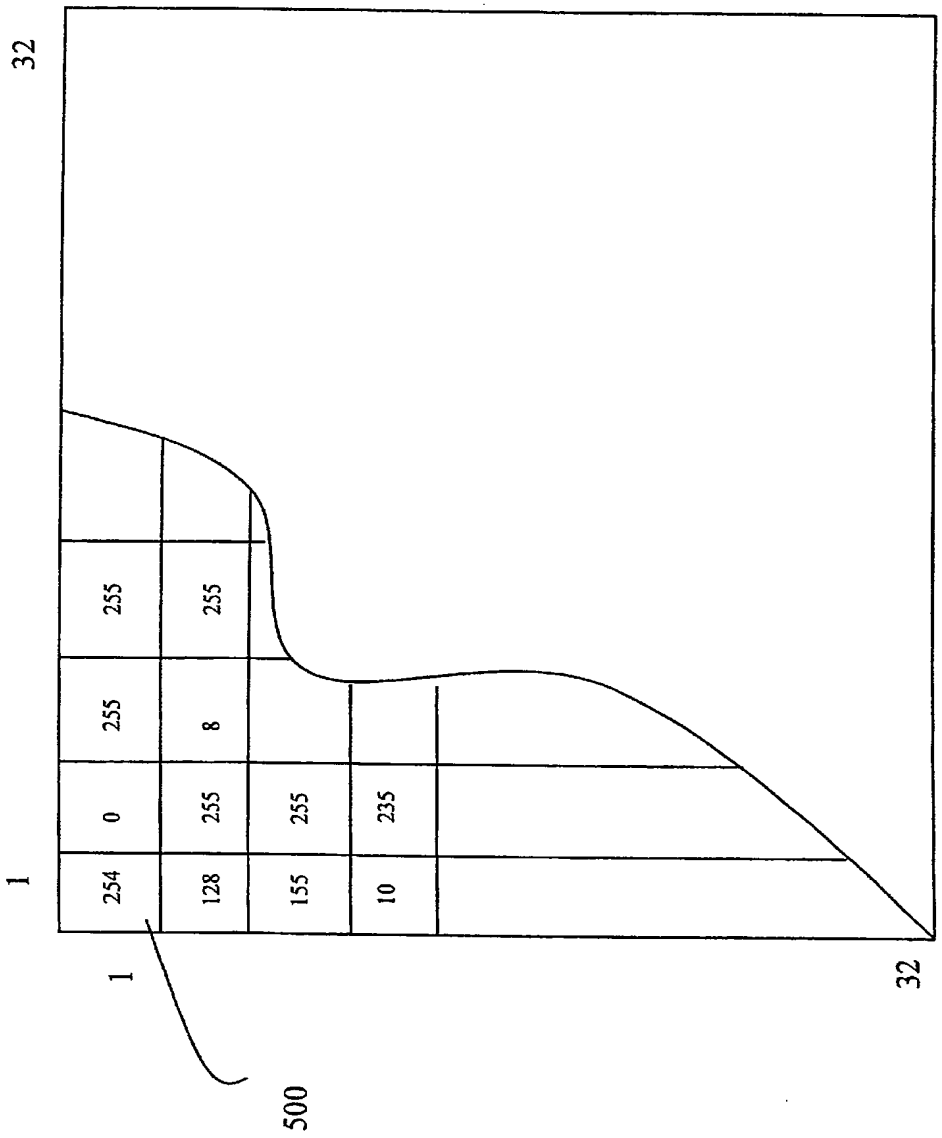


Fig. 5B

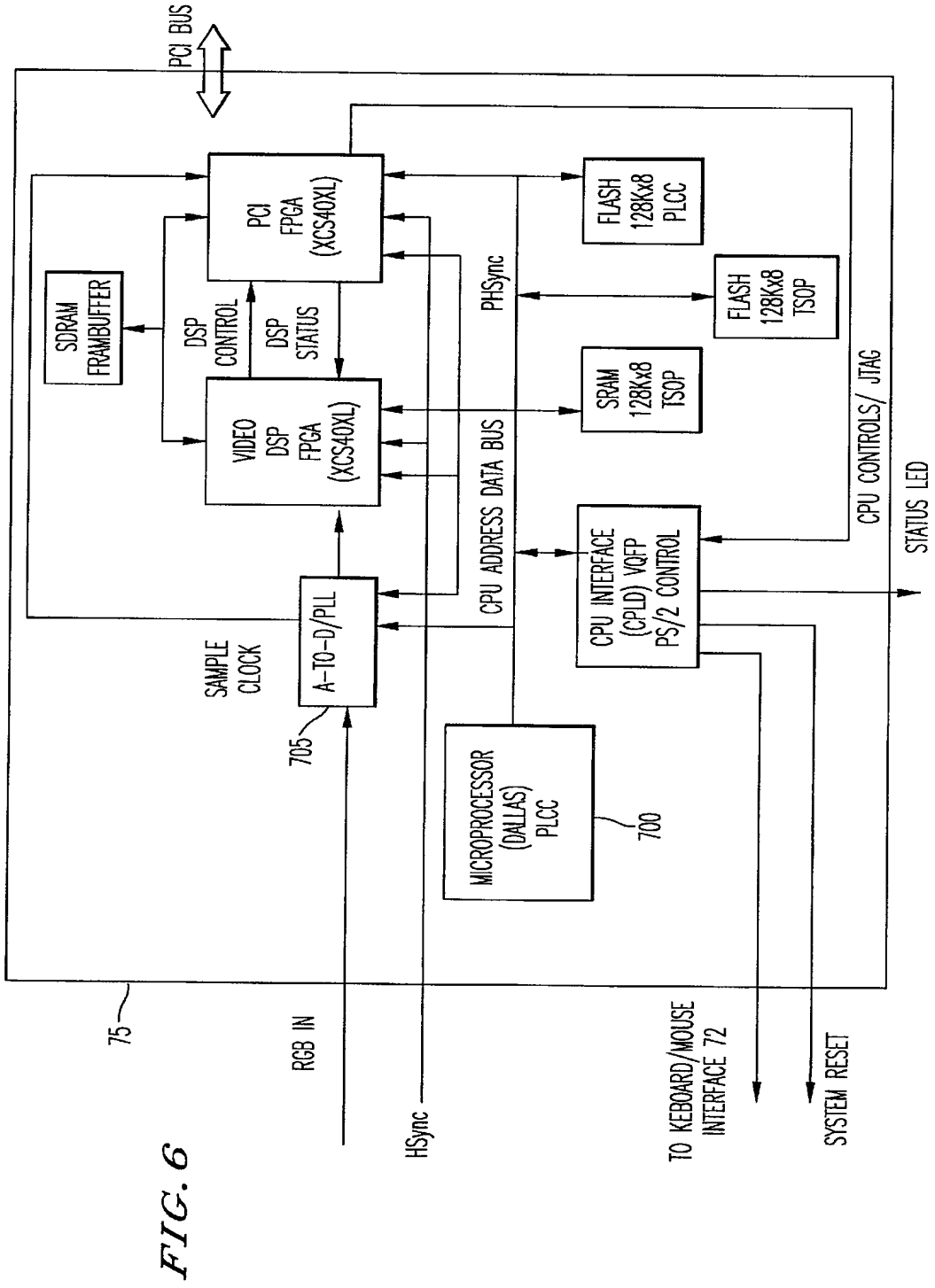

PCI BUS
PCI FPGA (XCS40XL)
FLASH 128Kx8 PLCC
FLASH 128Kx8 TSOP
SDRAM FRAMBUFFER
DSP CONTROL
DSP STATUS
PHSync
VIDEO DSP FPGA (XCS40XL)
SRAM 128Kx8 TSOP
CPU ADDRESS DATA BUS
CPU CONTROLS/ JTAG
CPU INTERFACE (CPLD) VQFP PS/2 CONTROL
STATUS LED
SAMPLE CLOCK
A-TO-D/PLL
705
MICROPROCESSOR (DALLAS) PLCC
700
75
RGB IN
HSync
TO KEBOARD/MOUSE INTERFACE 72
SYSTEM RESET

FIG. 6

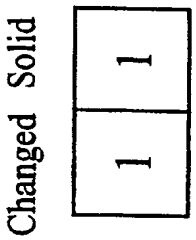


Figure 7a

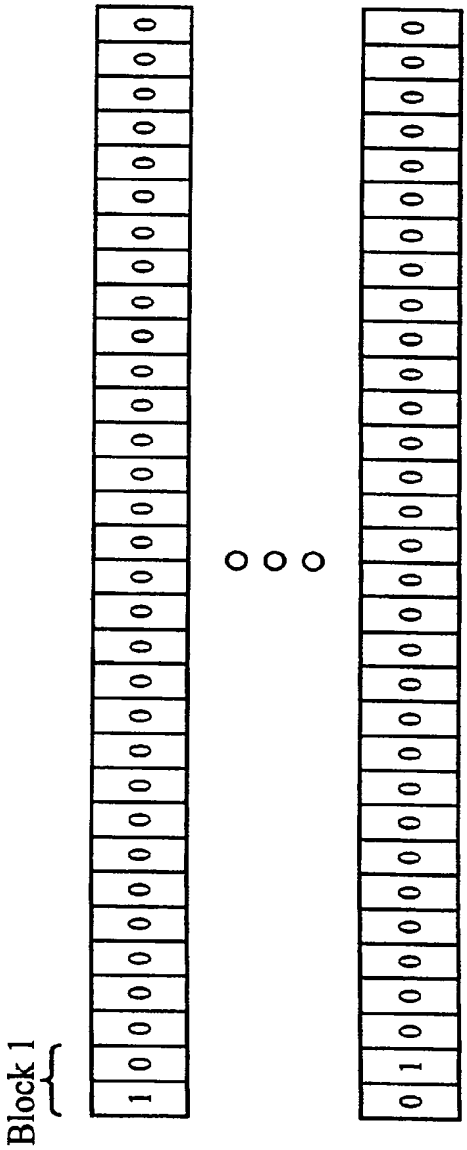


Figure 7b

Figure 8

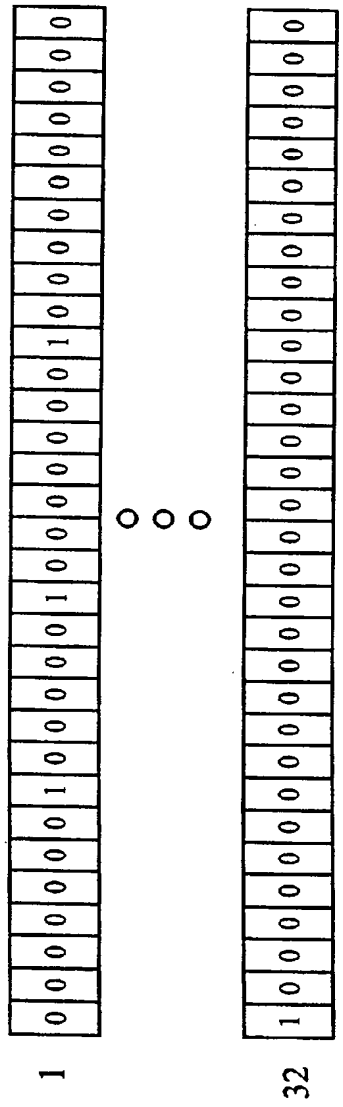


Figure 9a

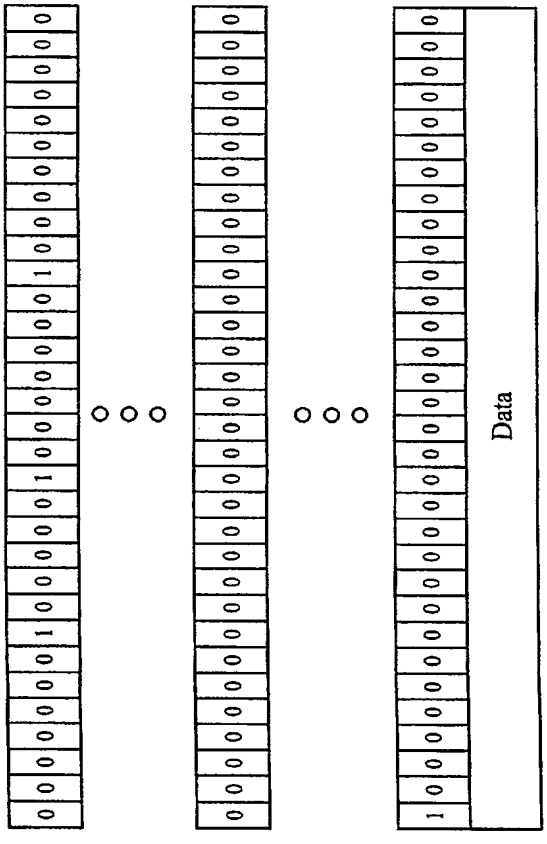
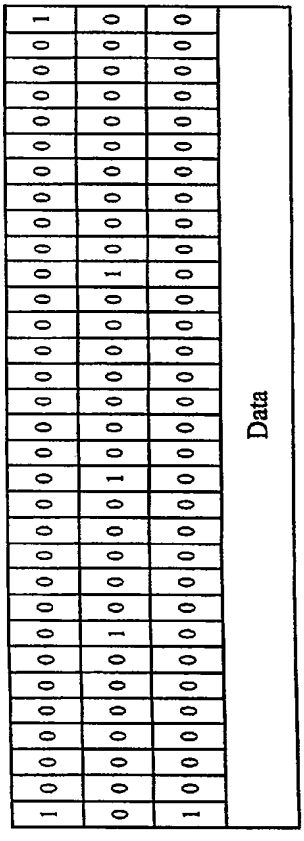


Figure 9b



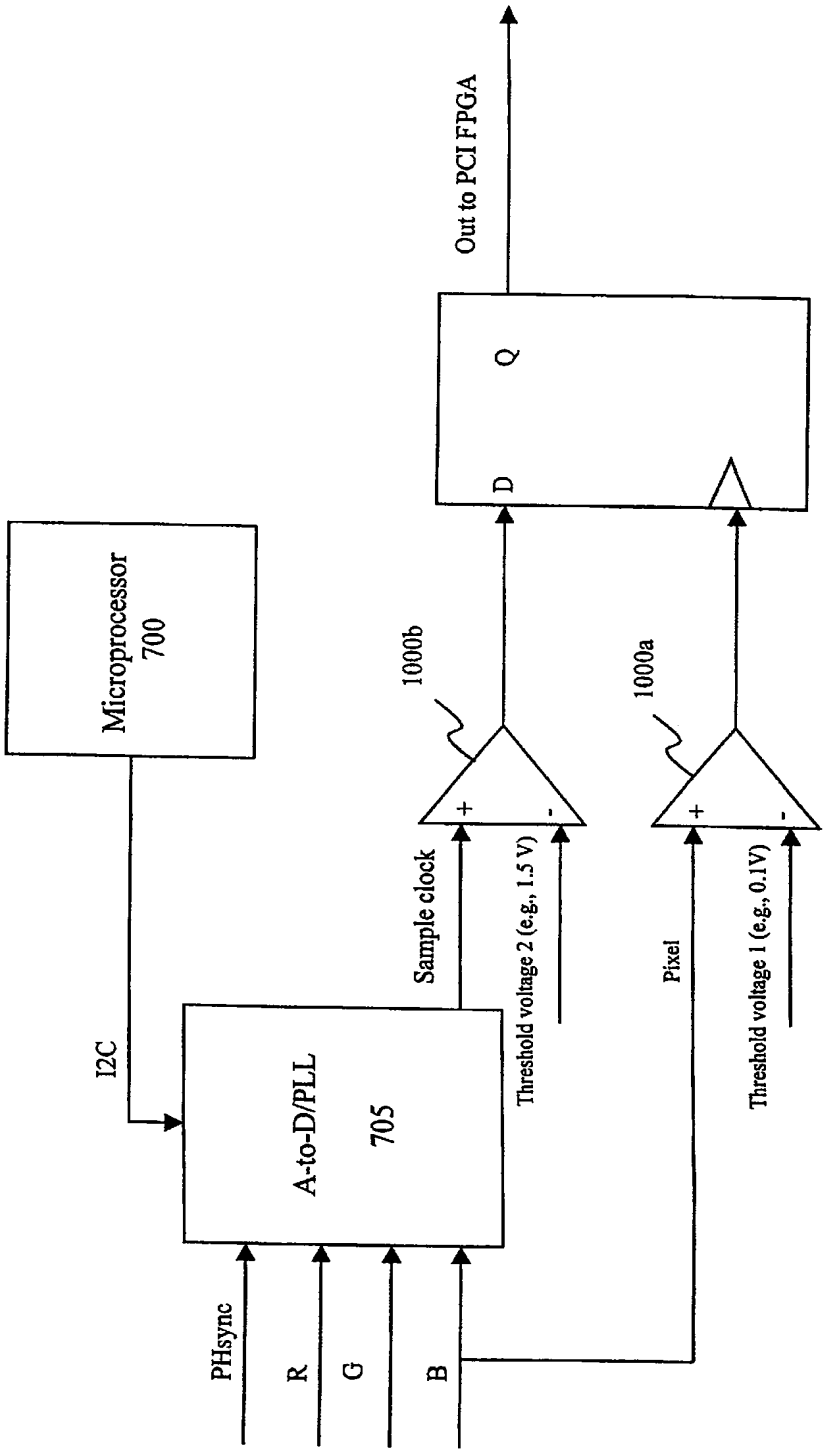

Out to PCI FPGA
Q
D
Microprocessor
700
1000b
1000a
+
-
+
-
Sample clock
Threshold voltage 2 (e.g., 1.5 V)
Pixel
Threshold voltage 1 (e.g., 0.1V)
I2C
A-to-D/PLL
705
PHsync
R
G
B

Figure 10

METHOD AND SYSTEM FOR INTELLIGENTLY CONTROLLING A REMOTELY LOCATED COMPUTER

CROSS-REFERENCE TO RELATED CO-PENDING APPLICATION

The present invention is a CIP of application Ser. No. 08/916,685, filed Aug. 22, 1997, now abandoned. The contents of that application are incorporated herein by reference.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention is directed to a method and system for intelligently controlling a remotely located computer. More specifically, the present invention is directed to a control system connected to a video output port and at least one data input port of a computer located in a first location. A user located in a second location, remote from the first location, controls the computer in the first location through the control system as if the user were directly connected to computer at the first location.

2. Discussion of the Background

Modem computing has migrated away from the use of centralized mainframes to the use of individual (or personal) computers. With that migration has come a decentralization of many of the resources that were centralized in a mainframe environment (e.g., peripheral devices including magnetic or optical disks and their associated files). That decentralization has not been accompanied by an equivalent increase in peer-to-peer networking capabilities such that those decentralized resources are available to a user as the user moves. Moreover, system administration of multiple physically remote systems increases maintenance concerns.

As a result of the lack of peer-to-peer access, a number of systems have been developed to provide control of remote computers. Unfortunately, many of those solutions have provided very limited control of the remote computer. The most rudimentary type of control is a text-based dialup connection. Control of the remote system is then performed through terminal emulation. Control using terminal emulation is also possible through network connections as opposed to dialup connections. Using (1) a telnet server (or daemon) on the remote computer and (2) a telnet client on the local computer, a user can connect to a remote computer—even across a wide area network (e.g., the Internet). However, telnet access also is limited by the fact that such control requires additional software (i.e., the server) to be running on the remote computer. Such server software may "crash" due to the errant operation of the computer. As a result, access to and control of the remote computer is lost after a crash or after a system "hang." In addition, such server software does not begin running on the remote computer until after the boot-up sequence. Thus, it is not possible to watch or alter the boot-up process using a telnet server.

More sophisticated remote control systems include the capability for graphics. Carbon Copy 32 from Compaq and LapLink from Traveling Software allow for remote access of computers while enabling a graphical user interface of the remote computer to be displayed at a user's local computer. Carbon Copy and LapLink on Windows 95, 98, NT and 2000 utilize "hooks" in the display subsystem of the remote computer to capture drawing requests (in the form of GDI calls). Those drawing requests are sent via a communications adapter to a Carbon Copy or LapLink client program running on the local computer. Once the drawing requests are received locally, the Carbon Copy or LapLink client program "re-executes" the requests so that the drawing operation is performed locally. Accordingly, the local computer displays both the local and remote images.

In addition, when using Carbon Copy or LapLink in a low to medium bandwidth connection (e.g., a 28.8 K or 56 K modem connection over a telephone line), the amount of data to be transferred becomes an important issue. In such a connection, there is insufficient bandwidth to send a complete copy of the screen frequently. PCAnywhere produced by Symantec of Cupertino, Calif. is an additional remote control program requiring server software on the remote computer in order to transfer graphics between computers.

An alternate graphical control system is the X Windows system, often run on UNIX workstations. Using X Windows, a server program running on a local computer receives drawing requests from an application running on (i.e., using the CPU and memory resources of) a remote computer. Although it is possible to utilize the X Windows graphical user interface over a wide area network, the X Windows system, like the terminal emulator and Carbon Copy systems, requires that application software be running on the remote computer in order to control the remote computer. That requirement prevents an X Windows-based system from being able to analyze or modify the boot process of the computers that it controls.

U.S. Pat. No. 5,732,212, to Perholtz et al., entitled "SYSTEM AND METHOD FOR REMOTE MONITORING AND OPERATION OF PERSONAL COMPUTERS," discloses a system in which the video, keyboard and mouse ports of a remote computer are connected to a host unit. The host unit may communicate with a local computer via a modem connection over phone lines. As described in the abstract of that patent, the video raster signal is converted to digital form.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide control of a remote computer independent of the operating system of the remote computer.

It is a further object of the present invention to provide a method and system for analyzing the screen information transmitted between the remote control system and the local computer in order to reduce the required bandwidth.

These and other objects of the present invention are provided by a remote control system that connects to a remotely located computer via a video port and one or more data input/output ports (e.g., keyboard, mouse, touchscreen). The system does not utilize resources of the remotely controlled computer, thus, the present invention operates independently of the operating system (and BIOS) of the remotely controlled computer.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete appreciation of the invention and many of the attendant advantages thereof will become readily apparent with reference to the following detailed description, particularly when considered in conjunction with the accompanying drawings, in which:

FIGS. 1A–1C are block diagrams of a system for accessing and controlling a remotely located target computer system according to the present invention;

FIG. 2 is a schematic illustration of the controlling computer of FIG. 1A;

FIGS. 3*a* through 3*c* are block diagrams of the relationship between the software and the hardware of several embodiments of the present invention;

FIG. **4** is a schematic illustration of a series of uncompressed video signals representing the image generated by the video card of the remote computer;

FIGS. **5**A and **5**B are graphical illustrations of the same block of the video memory of FIG. **4** between successive image captures by the system of the present invention;

FIG. **6** is a schematic illustration of one embodiment of an intelligent video digitizer as shown in FIG. 3*c*;

FIGS. 7*a* and 7*b* are block diagrams showing status registers indicating the status of blocks of the screen;

FIG. **8** is block diagram showing status flags indicating which bits in a block have changed;

FIGS. 9*a* and 9*b* are block diagrams showing compression headers and data for sending incremental changes; and

FIG. **10** is a block diagram of a circuit for altering the phase of when pixels are sampled.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring now to the drawings, in which like reference numerals designate identical or corresponding parts throughout the several views, FIG. **1**A is a block diagram of a system for accessing and controlling a remotely located computer system according to the present invention. In general, the system of the present invention transmits a GDI representation of digitized video signals as well as mouse and keyboard signals over a communications link. Since "local" versus "remote" is a matter of perspective, a set of consistent terminology is used throughout this application which ignores relative perspective. Herein, the phrase "target device" refers to a computer or switch that has its video output connected to the digitizer of the present invention. For example, in FIG. **1**A, the computers **20***a* through **20***c* are connected through switch **74***a*. Thus, any of those computers **20**, as well as the switch **74***a*, may be referred to herein as a target device. When referring to a target device that is a computer, that computer herein is referred to as a target computer. Similarly, when referring to a target device that is a switch, that switch is referred to herein as a target switch. Typically, the target computers are server computers that are connected to a computer network and operate to perform such tasks as controlling the operation of the network, storing commonly used programs or data, or connecting a local area network (LAN) to a wide area network (WAN) (e.g., the Internet). Those computers may be either computers in separate housings or part of a rack-mounted system. In an alternate embodiment, a target computer is a computer that controls any external hardware or equipment (including storage area network, factory equipment or consumer electronics/appliances).

By contrast, the computer that indirectly controls the target device(s) is referred to herein as "the controlling computer." The computer **12** in FIG. **1**A is the controlling computer and is shown in greater detail in FIG. **2**. Specifically, the computer **12** includes a computer housing **102** that houses a motherboard **104**. The motherboard **104** includes a CPU **106** (e.g., Intel 80x86, Motorola 68x0, or PowerPC), memory **108** (e.g., DRAM, ROM, EPROM, EEPROM, SRAM, SDRAM, and Flash RAM), and other optional special purpose logic devices (e.g., ASICs) or configurable logic devices (e.g., GAL and reprogrammable FPGA). The controlling computer **12** also includes plural input devices, (e.g., a keyboard **122** and mouse **124**), and a display card **110** for controlling monitor **120**. In addition, the computer system **12** further includes magnetic or optical storage devices. Such storage devices include, but are not limited to, a floppy disk drive **114**; compact disc reader **118**, tape; and a hard disk **112**, any of which are connected using an appropriate device bus (e.g., a SCSI bus, an Enhanced IDE bus, or an Ultra DMA bus). Also connected to the same device bus or another device bus, the computer **12** may additionally include a compact disc reader/writer unit (not shown) or a compact disc jukebox (not shown). Although a compact disc **119** is shown in a CD caddy, the compact disc **119** can be inserted directly into CD-ROM drives that do not require caddies. In addition, a printer (not shown) also provides printed listings of operations of the present invention.

As stated above, the system includes at least one computer readable medium. Examples of computer readable media are compact discs **119**, hard disks **112**, floppy disks, tape, magneto-optical disks, PROMs (EPROM, EEPROM, Flash EPROM), DRAM, SRAM, SDRAM, etc. Stored on any one or on a combination of computer readable media, the present invention includes software for controlling both the hardware of the computer **12** and for enabling the computer **12** to interact with a human user. Such software may include, but is not limited to, device drivers, operating systems and user applications, such as development tools. Such computer readable media :frther includes the computer program product of the present invention for remotely accessing and controlling a target computer (or switch). The phrase "computer code devices" as used herein can be either interpreted or executable code mechanisms, including but not limited to scripts, interpreters, dynamic link libraries, subroutines, Java methods and/or classes, and partial or complete executable programs. Moreover, although portions of the specification describe the operation of portions of the present invention in terms of a microprocessor and a specially programmed memory, one of ordinary skill in the art will appreciate that a portion of or all of those described functions may be implemented in a configurable logic device. Such a logic device may be either a one-time programmable (OTP) logic device or a field programmable gate array (FGPA). It will also be appreciated by one of ordinary skill in the art that a single computer code device and/or logic device may implement more than one of the described functions without departing from the spirit of the present invention.

In addition, in a first embodiment using a "system on a chip," the present invention is implemented as (1) a digital system that includes an integrated microprocessor, memory and specialized logic on a single- or multi-chip module and (2) analog-to-digital and digital-to analog converters. In a second embodiment using a "system on a chip," the present invention is implemented as a mixed-signal system that includes an integrated microprocessor, memory, specialized logic, and analog-to-digital and digital-to analog converters on a single or multi-chip module. As used herein, "means" will be understood to include any one of the computer code devices, logic devices, and/or systems on a chip, in any combination. That is, although one "means" may be a computer code device, it may interact with another "means" that is a logic device.

The controlling computer **12** also includes a communications device **53** for communicating with the target device(s). Such a device **53** may include (1) a modem for connecting via a telephone connection, (2) a wireless transceiver for wirelessly communicating, and (3) a wired adapter (e.g., an

Ethernet or token ring adapter). In any of those configurations, the controlling computer **12** communicates with a target controller **50** using any selected communications protocol (e.g., TCP/IP, UDP, or RDP). In an alternate embodiment, the controlling computer **12** is a set-top box that receives the output of a target device via a television connection (cable or satellite) and enables the output to be displayed on a television or similar device (e.g., WebTV). Controlling computer **12** can likewise be a notebook, hand-held or palm-top computer.

In addition, more than one communications device **53** can be used simultaneously. For example, two or more communication devices may be combined in parallel in order to increase bandwidth. Moreover, separate adapters may be used for transmitting and receiving. Moreover, although the controlling computer **12** is illustrated as using a single communications channel, in an alternate embodiment, plural communications channels are used to communicate with plural independent target computers.

Commands or keystrokes entered at the keyboard **122** or mouse **124** of the controlling computer **12** operate to control the target computer **20** as if the command had been entered using a keyboard or mouse that is directly connected to the target computer **20**. In addition, the monitor **120** of the controlling computer **12** displays the same video signals that are captured from the video adapter of the target computer **20**.

Generally, a target controller **50** is a computer including at least one controller card. Each controller card is connected to one or more target devices (i.e., computer **20** or switch **74**). Each controller card physically connects to at least one set of interfaces including: (1) a video interface **82**, (2) a keyboard interface **84** and (3) a mouse interface **86**. In an alternate embodiment, the keyboard and mouse are merged into a single interface (e.g., USB or Macintosh-style). (In an alternate embodiment, one or more interfaces may be wireless, and "connected peripheral devices" as used herein shall refer to wired and wireless peripheral devices.)

In addition, the total number of target devices that are logically connected simultaneously may be even greater if the target device is a switch **74***a* (connected to several target computers **20**) rather than a single target computer **20**. Moreover, although each of the target computers **20***a* through **20***c* is illustrated as having separate housings, the present invention is not limited thereto. More than one target computer may be contained within a single case.

In the embodiment shown in FIG. **1**A, the target controller **50** is implemented as a computer having similar components to the controlling computer **12**. Those components include computer code devices for performing portions of the method of the present invention. In the embodiment of FIG. **1**A, the target controller **50** includes at least one internal "plug-in" or "add-in" card labeled "Controller card **1**." In an alternate embodiment, the target controller **50** includes at least one controller integrated onto the motherboard of the computer. In either of those embodiments, the target controller **50** optionally also attaches to local keyboard, mouse and video connections.

In yet another alternate embodiment, the target controller is a stand-alone device similar to a router or a switch. In the router/switch configuration, the keyboard, mouse and screen are not required and the router/switch is configured remotely—either through the communications device **53** or through a separate control interface (not shown). Remote configuration may be via a direct connection, a local area network or a wide area network (e.g., the Internet). In addition, the router/switch configuration may be updated through a removable medium (e.g., a floppy disk or CD-ROM) inserted into the router/switch. In the preferred embodiment, the target controller **50** is a computer system running Windows NT (or its successor Windows 2000) and is connected to at least one plug-in card. Alternate embodiments utilize Windows CE, UNIX, Linux or MacOS as the operating system.

The target controller **50** can be further reduced and integrated into a KVM switch or into another target device (e.g., integrated on the motherboard of a target computer or included on a peripheral card of the target computer). Illustrative embodiments are shown in FIGS. **1**B and **1**C.

After configuration, the target controller **50** operates to capture the video output of the target device. The captured video signals are stored in either a frame buffer internal to the controller card or in a memory shared with other components of the computer. In addition, the controller card **50** fills a set of keyboard/mouse buffers internal to the controller card with keyboard and mouse commands to be sent to the target device. If the target device supports bidirectional mouse and keyboard communication, then the controller card also includes at least one buffer for receiving communications from those devices. Those commands are sent to the controlling computer **12**.

The controller **50** includes a video digitizer that receives and converts the analog signals output by connected target device. The controller stores the converted signals in digital form in the video memory (shared with the mother board or dedicated to the controller card) as digital video data. After a configurable amount of processing, the digital video data is sent from the target controller **50** to the controlling computer **12**. Based on the desired cost, complexity and performance of the controller, various processing tasks are divided between the hardware and software of the controller **50**.

The initial starting point, however, is the pixel depth of the pixels to be rendered on the controlling computer **12**. In order to determine that depth, a user must consider both the depth of the target device and the amount of available bandwidth between the controller **50** and the controlling computer **12**. If the pixel depth being transferred is low but the pixel depth of the target device is high, then the ability to represent color gradations may be severely impaired. In fact, similar colors that are readily distinguishable on the target device may become indistinguishable on the display of the controlling computer. On the other hand, higher pixel depths require larger amounts of bandwidth to transfer and some loss of color separation may be acceptable.

In one embodiment of the present invention, the controller samples at eight bits per color, providing a 24-bit color sample. In another embodiment, the present invention samples at 5 bits per color to reduce the cost of the A/D converter. The samples are then converted into a bitmap in one of several formats: (1) 8-bits-per-pixel, (2) 16-bits-per-pixel, (3) 24-bits-per pixel, and (4) device independent.

FIG. **3***a* illustrates that, in a first embodiment, the hardware (digitizer **70**) and software **230** (including device driver **210** and the digitizer control application **220**) of the controller **50** simply act as a thin interface to a remote control software application **200**. When providing the thin interface, neither the software **230** nor the digitizer **70** performs any analysis on the video signals captured by the digitizer **70**. Instead, the digitizer control application **220** periodically requests (through the device driver **210**) that a whole screen of data be sampled. The digitizer control

application **220** then draws the whole captured screen to its local screen using Windows GDI calls. The remote control software application **200** captures those GDI requests and retransmits them to the controlling computer **12**. The client software on the controlling computer **12** then re-executes the commands so that the screen of the controller **50** and the screen of the controlling computer **12** show the same image.

An illustration of an exemplary method of storing the captured/digitized data is shown in FIG. **4**. In that illustration, the red, green and blue components of each pixel are captured and stored together. In an alternate embodiment, the red, green and blue values are stored separately such that the red value for pixel number **1** is adjacent in memory to the red value for pixel number **2**.

Several embodiments are possible for the storage and transmission of the digitized data. It is possible that the data is quantized at one depth (bits-per-pixel), stored at a second depth (greater or less than the quantized depth), and transmitted in a third depth. However, in an alternate embodiment, one or more of those depths may also be the same. In the case of quantizing at 5 bits per color (i.e., 15 bits per pixel), the 15 bits per pixel are converted into a device independent bitmap using 24 bits per pixel. Prior to transmission by LapLink or Carbon Copy, the 24 bits per pixel are converted to a "closest" color in the corresponding color palette (which may be 8 bits per pixel).

Although compression is not required, in this thin-interface embodiment, the preferred remote control software application **200** is LapLink by Traveling Software since, before transmission to the controlling computer **12**, LapLink performs some analysis and lossless compression on the image resulting from the captured GDI calls. Accordingly, in that thin-interface embodiment, LapLink can be replaced by any other remote control application but preferably one that also performs lossless compression on the captured GDI calls before transmission.

In the second embodiment illustrated in FIG. **3***b*, the digitizer **70**, the device driver **210**, and the remote control software **200** remain consistent with their corresponding parts described in relation to FIG. **3***a*. However, the digitizer control application **220** of FIG. **3***a* is replaced by an analyzing digitizer control application **240**. The analyzing digitizer control application **240** requests, through the device driver **210**, that a screen be captured (i.e., digitized). Rather than using GDI calls to redraw the entire screen (which would be captured in its entirety by the remote control software **200**), the analyzing digitizer control application **240** analyzes the captured image and uses GDI calls to redraw only changed blocks instead. Those changed blocks are captured by the remote control software **200**.

For example, in the preferred embodiment of this implementation, the analyzing digitizer control application **240** partitions a screen into blocks (e.g., 32 pixels by 32 pixels), an example of which is shown in FIG. **5***a*. Although one embodiment uses fixed size blocks, an alternate embodiment uses blocks of varying size and shape. For example, where large blocks of the screen are a single color, the block size may be increased (e.g., to 64×64 or 128×32) in order to optimize solid block transmission, as is described in greater detail below. Although any size block can be used, other preferable blocks size are: 16×16, 16×32, 32×16, and 64×16.

For each block, the analyzing digitizer control application **240** determines if there is a more efficient way to draw a block. One method of drawing a block utilizes identification of solid blocks—i.e., blocks of a single color. In many backgrounds, there exist regions that are a single color (e.g., all blue or all white). Once identified, those blocks can be more efficiently drawn by using a single GDI call indicating that a colored region is to be drawn at a particular (x, y) location on the screen. This method, however, requires that the CPU of the computer system perform the analysis of which blocks are a single color. In a high resolution, 1280×1024 screen using 32×32 blocks, for each screen update, the CPU checks 1280 blocks that are 32×32 pixels each.

The present invention may also identify "solid" blocks which are blocks that probably should have been a single color, but, through errors in digitization, are not exactly one color. The present invention can be configured to establish (1) a percentage threshold, (2) an intensity threshold or (3) both. The percentage threshold represents the number of errant pixels within a block that can deviate from the "solid" color, regardless of how far from the "solid" color they are, and still treat the block as a solid block. The intensify threshold represents the amount that any pixel can vary from the "solid" color before the block is considered not to be solid. By combining the percentage threshold and the intensity threshold, the system can limit both the number of errant pixels and amount of variation, simultaneously.

Improved performance is not, however, limited to identifying solid-colored blocks. The analyzing digitizer control application **240** can also improve efficiency by tracking which blocks change between successive screen captures. To track those changes, the analyzing digitizer control application **240** double buffers the digital video information received from the device driver. In this way, the analyzing digitizer control application **240** can compare (1) the screen information stored in a first buffer for a previous frame and (2) the screen information stored in a second buffer for the image currently being captured. The buffer sizes need not actually be the same sizes as long as the corresponding blocks can be compared in a non-destructive fashion such that the currently captured block can replace the corresponding block from the previous screen after comparison. Having identified the changed blocks, the analyzing digitizer control application **240** then need only redraw the changed areas as they change. The remote control software **200** then captures and transmits those changed blocks.

Unfortunately, as described above, the digitization/quantization process may introduce errors in producing digital data. Those errors not only affect the ability to identify solid blocks, those errors also cause blocks to appear as if they changed when the blocks have actually remained constant. For example, the memory block shown in FIG. **5A** represents the data sampled during a first time period. The memory block shown in FIG. **5B** represents the same block sampled during a subsequent time period. As can be seen, the value in location **500** has changed from **255** to **254**. Without further analysis, it would appear that this block has changed. In the illustrated example, the change requires that the block be retransmitted. In all likelihood, the value would change back a short time later and the block would be retransmitted yet again.

To prevent such digitization errors from increasing the amount of data transferred between the target controller **50** and the controlling computer **12**, in one embodiment of the analyzing digitizer control application **240**, the analyzing digitizer control application **240** filters the sampled data to hide small changes. In a first filtering embodiment, the analyzing digitizer control application **240** stores both the filtered data from a previous image and an unfiltered copy of the previous image. The current image is then captured, stored and a filtered version of the current image is stored

separately from the unfiltered version. (It will be appreciated by one of ordinary skill in the art that the entire current image and its filtered equivalent need not be stored. Rather, once the processing of a block (or group of blocks) is complete, the previous block is replaced by the current block, and the area for the "current" block is reused for the next block.)

In one embodiment, a finite impulse response (FIR) filter averages the current pixel's value and the pixel value from the previous frame. That average is then averaged with the previous average from the previous frame. (Rounding (up or down) may be used in light of the division that is inherent in the averaging process.) The two filtered images are compared for changes. If there are changes, then the block is drawn, in either its filtered form or its unfiltered form.

In another filtering embodiment, the analyzing digitizer control application **240** stores a copy of the unfiltered block for a previously sampled screen and calculates differences between the unfiltered block and a currently sampled block. The differences are stored in a difference block, and the difference block is filtered and compared against a threshold (or compared against a threshold and then filtered) to determine if the new block (or portions thereof) should be redrawn. (It will be appreciated by one of ordinary skill in the art that the filtering step may be omitted if the use of a threshold is found to be sufficient to avoid quantization errors.)

In any of the above filtering embodiments, the analyzing digitizer control application **240** may actually inadvertently prevent small changes from being transmitted to the controlling computer **12**—even when the changes are the result of an application's actions. To prevent the filtering and thresholding from impeding a user's ability to see those small changes, blocks that have changed (but that nonetheless have changed less than the threshold amount before or after filtering), may be sent (in whole or in part) when bandwidth is available. An area of interest may also be designated by the user such that these system ignores changes to sampled data in the area outside of the area of interest.

In one embodiment of the present invention, the filtering of blocks is changed dynamically. For example, the threshold levels may be increased when the user wants to decrease network traffic. In addition, in an alternate embodiment, the system includes a percentage threshold that causes a block not to be treated as changed as long as a total number of pixels within the block that have changed is less than the threshold—regardless of how much those pixels have changed. As a result fewer blocks are treated as "changed" and fewer drawing requests are made. Likewise, the system may change from one block size to another or from one filter to another.

The filtering and thresholding process described above with reference to the analyzing digitizer control application **240**, may likewise performed (wholly or partially) in hardware as part of an intelligent digitizer **75** shown in FIG. **3***c*. The intelligent digitizer **75** is shown in greater detail in FIG. **6**. The video A-to-D/PLL **705** is a triple high speed Analog-to-Digital Converter that contains an integrated PLL, and a serial digital interface for setting individual registers (e.g., registers controlling control the pixel clock and clamping settings). The input signal used by the PLL is the polarized HSYNC (PHSync) signal. This is then multiplied by the value set in one of the internal registers to produce the desired pixel clock frequency. The output is then provided to the Video DSP and PCI FPGAs in order to capture video at the required pixel clock rate.

In one embodiment of the present invention shown in FIG. **10**, the system adjusts when the pixel is sampled by adjusting the phase of the A-to-D convertor **705**—i.e., the delay between the active edge of the PHSYNC signal as compared to the first active edge of the sample clock after the active edge of the PHSYNC signal. As shown in FIG. **10**, in the preferred embodiment, the blue signal from the RGB inputs is used as the positive input to the comparator **1000***a*. In alternate embodiments, the red or green signal may be used. In yet another embodiment, two or more of the color signals are combined to form the positive input. As shown in FIG. **10**, the blue signal is filtered by applying a low threshold signal to the negative input of the comparator **1000***a*. The filtered blue signal then acts as the clock input of a D flip-flop **1005**. The output of the A-to-D converter **705** is the sample clock shown in FIG. **6**), which is also applied to the D input of the D flip-flop **1005**. The output of the D flip-flop ig fed out to the PCI FPGA where its status can be read by the analyzing control application **220** as if the output were part of a register of the FPGA. In the preferred embodiment, the D flip-flop is included in the CPU Interface CPLD.

In order to control the phase, the analyzing control application **220** reads the status of the output of the D flip-flop **1005** (e.g., once per frame). When the output is a 1, the delay of the A-to-D convertor **705** is moved one unit in a first direction by sending a command to the microprocessor **700** (which then adjusts the delay using the I2C bus). Conversely, when the output is a 0, the delay is moved one unit in a second direction opposite the first direction. In the A-to-D convertor **705** of the preferred embodiment, each unit corresponds to approximately 11 degrees. In light of this circuit and the fact that the delay is reprogrammed, the system will oscillate between reading a status of 1 and 0. This causes the beginning of pixel data to correlate with the trailing edge of the sample clock signal, As such, the next rising edge of the sample clock signal will be at the center of the period in which the blue signal (and the red and green signals) hold valid data.

In an alternate embodiment, additional smoothing logic (either hardware or software) is used to slow down the changes in phase. Rather than toggling between shifting forward and shifting backward, at each sample, the logic can decide to forego a change after a status read. In order to decide when to change, a running average (or other filtering function) can be used to determine the effect of changing or not changing.

The A-to-D/PLL also has a number of internal registers that allow the board to have control over the phase relationship of the input signals and the output clock signal. This allows adjustments to be made on the sampling clock to ensure that the input signal is sampled on the optimal location and minimize jitter caused by sampling during transition. It also has settings for adjusting the voltage level offset and gain to allow for adjustment due to level shifting and attenuation over the video cable. In the preferred embodiment, the A-to-D/PLL is the Philips TDA8752H/8—a triple high-speed (100 MHz) analog to digital converter. It contains all of the phase-locked-loop circuitry necessary to generate the pixel clock from the Horizontal Sync signal. The TDA8752 has numerous control registers that are set by the microcontroller via an **12**C interface.

One set of possible resolutions that can be used by the present invention is shown in Table I below.

TABLE I

COMMON VIDEO MODES DEFINED

Resolution	Vert. Freq.	Horiz. Freq.	Lines/Frame	Pixels/Line	H/V Level	PCLK Modulus	PCLK Freq.
DOS	70 Hz	31.5 KHz	450/???	???/???	LOW/HIGH		
640x480-60	60 Hz	31.5 KHZ	480/525	640/800	LOW/LOW		25.175 MHz
640x480-72	72 Hz	37.9 KHz	480/520	640/832	LOW/LOW		31.500 MHz
640x480-75	75 Hz	37.5 KHz	480/500	640/840	LOW/LOW		31.500 MHz
640x480-85	85 Hz	43.3 KHz	480/509	640/832	LOW/LOW		36.000 MHz
800x600-56	56 Hz	35.1 KHz	600/625	800/1024	HIGH/HIGH		36.000 MHz
800x600-60	60 Hz	37.9 KHz	600/628	800/1056	HIGH/HIGH		40.000 MHz
800x600-72	72 Hz	48.1 KHz	600/666	800/1040	HIGH/HIGH		50.000 MHz
800x600-75	75 Hz	46.9 KHz	600/625	800/1056	HIGH/HIGH		49.500 MHz
800x600-85	85 Hz	53.7 KHz	600/631	800/1048	HIGH/HIGH		56.250 MHz
1024x768-60	60 Hz	48.4 KHz	768/806	1024/1344	HIGH/HIGH		65.000 MHz
1024x768-70	70 Hz	56.5 KHz	768/806	1024/1328	HIGH/HIGH		75.000 MHz
1024x768-75	75 Hz	60.0 KHz	768/800	1024/1312	HIGH/HIGH		78.750 MHz
1024x768-85	85 Hz	68.7 KHz	768/808	1024/1376	HIGH/HIGH		94.500 MHz

As would be appreciated by one of ordinary skill in the art, other resolutions are possible. The determination of other possible modes may be aided by reference to VESA and Industry Standards and Guidelines for Computer Display Monitor Timing, Version 1.0, Revision 0.7, Revision Date: Dec. 18, 1996, the contents of which are incorporated herein by reference.

In addition to the above factors used to control video modes, the system of the present invention also controls when sampling begins following an (P)HSYNC signal or a VSYNC signal. The time from signal to first sample is called the "front porch." If sampling after an (P)HSYNC signal begins too early (i.e., the front porch is too short), the system will sample "black" pixels prior to the real left edge of the display. If sampling after an HSYNC signal begins too late (i.e., the from porch is too long), the system will miss sampling the beginning pixels of the display. Similar problems exist for timing with relation to the VSYNC signal. Accordingly, the present invention provides the ability to set the front porch.

In one embodiment of the present invention, the front porch is set manually through user intervention—typically through a trial and error process. In three automated embodiments, the system of the present invention provides automatic determination of the front porch when a non-black background is used. In the first automated embodiment, the right edge of the screen is used as a reference. Thus, the system uses an initial front porch value, counts out the number of pixels in a row, and then determines if the pixel after the end of the row is black or colored. If that pixel is black using the initial front porch value, then the front porch value is shortened and the counting process is repeated. This shortening process is repeated until a non-black pixel is found in iteration I. Then the front porch value is reverted to the front porch value in iteration I–1—i.e., to the front porch value in the previous iteration. On the contrary, if the pixel is colored when using the initial front porch value, then the front porch value is increased until a black pixel is found at the end of a row in iteration I. The front porch value is then reverted to the delay value in iteration I–1—i.e., to the front porch value in the previous iteration.

In the second automated embodiment, a process similar to the first automated embodiment is used, except that the beginning of the row is analyzed. If the beginning of the row is found to be black, then the front porch value is increased until a non-black pixel is found in iteration I. Conversely, if the beginning of the row is found to be colored, then the front porch value is decreased until a black pixel is found in iteration I. Then the front porch value is reverted to the front porch value in iteration I–1.

In a third automated embodiment, the processes of the first and second automated embodiments are combined—thereby checking the left and right edges. In this manner, the correct number of pixels per line can also be automatically determined. A similar process can be performed for the vertical delay looking at (1) the top row, (2) the bottom column, or (3) the top and bottom columns.

The Flash memory component(s) contains all non-volatile data required to enable the onboard microprocessor to control operation of the intelligent digitizer **75**. Flash information includes: (1) Microprocessor Program/Backup/Boot code and (2) a PCI FPGA Initialization Bitstream. If sufficient free memory space exists on the Flash, then the Flash also contains backup copy of the last correctly programmed PCI FPGA Initialization Bitstream. This enables the digitizer **75** to be reloaded in case of an error in programming. One embodiment of the Flash configuration uses one PLCC Flash device with a TSOP Flash device soldered on the board.

In one embodiment of the Flash memory device, the memory is physically addressed as a single large memory device. In an alternate embodiment, the memory is physically divided into pages that can be used as the microprocessor decides. By setting the page bits in a page register, the system can change from one page to another. For example, using two page bits, **00**=page **0**, **01**=page **1**, **10**=page **2**, and **11**=page **3**. As the number of page bits increases, the number of independently addressable pages increases. This aids in providing a larger accessible memory to those microprocessors that have small address bus sizes.

The SRAM component contains both User Data to be used for general purpose RAM and program data when the microprocessor needs to run the program from RAM. In one embodiment of the SRAM memory device, the memory is physically addressed as a single large memory device. In an alternate embodiment, the memory is physically divided into pages as described above.

The CPU Interface CPLD is intended to provide all of the CPU's address/data bus interfacing signals including the chip selects to memory, the FPGA, and any external signals that need to be read from MMIO. By way of a non-limiting example, the FPGAs, CPLD and SDRAM run off a 3.3 volt power supply. The other components may use the same or different supply voltages.

The PCI FPGA provides the communication interface between the CPU of the computer and the local microprocessor **700** onboard the controller **50**. Thus, the PCI FPGA receives requests sent by the device driver **210**. It also provides access to the video buffer and supporting registers (e.g., bit change, block status). Although depicted as an FPGA, one of ordinary skill in the art would appreciate that the communication interface also can be either an application specific integrated circuit (ASIC) or a one-time programmable (OTP) circuit if the interface does not need to be field updated. The interface provides the following features (through the device driver **210**): (1) re-programming the CPLD over a JTAG interface; (2) detecting video presence; (3) detecting video resolution parameters; (4) intializing the frame buffer; (5) polarizing sync signals; (6) controlling the Video DSP FPGA; (7) resetting the components of the controller **50**; and (8) setting the active video parameters.

The Video DSP FPGA performs most of the video signal processing required to capture, filter, detect changes in frames, and store the video in a frame buffer (e.g., a SyncDRAM memory device). The PCI FPGA controls operation of the video DSP including any modes that the video DSP has for capturing video.

By providing separate programming interfaces for the two FPGAs, the video DSP FPGA can be updated without reprogramming the PCI interface that interfaces directly with the PCI bus.

The microprocessor of the controller controls most of the local data flow on the controller **50**. That microprocessor performs: (1) Basic system testing (e.g., code checking, FPGA checking, and RAM testing), (2) transferring mouse and keyboard signals, (3) downloading new programs or FPGA boot code; (4) initializing the onboard FPGAs; and (5) communicating with the analog-to-digital converter to control pixel clock settings (e.g., phase and frequency) and video settings (e.g., color offsets). The microprocessor may act as a watchdog timer to ensure that the system is running properly. If the system is not running properly, the microprocessor can then reset the system.

When the controller is first powered on, a power-on reset is performed internally by the CPU. (The RESET pin is held low at power-up by a pull-down resistor until the FPGA is booted. Once booted, the FPGA will drive the signal low unless a reset is asserted by the application). Later, the controller **50** may be reset by receiving a command from the communication interface. This signal forces a hardware reset to the microprocessor and resets the CPU and all registers to a known state. The controller **50** may be partially or completely reset by using commands to perform: (1) a CPU reset, (2) a CPLD reset, or (3) a video DSP reset. The CPU Reset resets the CPU and the CPLD interface logic to the CPU. This allows the application to set the CPU and any logic that will affect the operation of the CPU to a known and initial state. In addition, the CPU may have the capability through independent logic to cause a self-reset.

The CPLD reset resets the additional circuitry that does not interface to the CPU. The logic that allows the CPU to reset itself functions independently from the interface logic. In addition, the Video DSP Reset allows either the application or CPU to reset the internal logic of the Video DSP FPGA to either recover from a locked-up or to re-initialize any internal logic that needs to be set to a known state. Preferably, all of the reset signals are active high and are tri-stated with a pull-down resistor. This allows multiple sources to signal a reset without causing contention. An active high reset provides consistency with the CPU's reset polarity.

When the controller **50** determines that the target device is a target switch rather than a target computer, the controller can provide additional functionality specific to the switch. The controller can provide "thumbnail" images of target computers connected to a target switch to allow many target systems to be displayed at the same time, shown in miniature.

The control applications (**220** and **240**) utilize a multi-window architecture (e.g., the Multiple Document Interface (MDI)) to support control for multiple target devices. When a target computer's window gains focus, the target controller **50** automatically sends the appropriate keystroke sequence (e.g., "<PrtScr>+number+<Enter>") to the switch to select the corresponding switch port of that target computer. When the mouse and keyboard have been inactive for a specified time interval, the controller will optionally enter a scan mode. In this mode, the target-system windows are updated in a repeating sequence. To update each of the target computers, the controller card sends a switch command (i.e., a keystroke sequence (e.g., "<PrtScr>+number+<Enter>")) to select the next target device. The video output of that target device is then sampled, and the sampled image is written using GDI calls. Any mouse or keyboard activity cancels scan mode, and only the selected target window continues to be updated.

In one embodiment of the system of the present invention, the user (with the help of a configuration file or configuration "wizard") manually establishes the correlation between the name of a system and its switch/port number. In light of the fact that this manual process can be cumbersome, especially when switched are tiered in a hierarchy, an alternate embodiment utilizes an automated configuration process. In that embodiment, the switches utilize one of the keyboard or mouse ports or a separate dedicated communications port to pass information from the target devices or switches up to the target controller **50**. In yet another embodiment, the target controller **50** receives configuration information from a network computer about the port/switch configurations.

In a more secure embodiment, the present invention includes security features to restrict the computers that can be viewed or accessed (or both) by the remote control software. For example, using this security, one user may only be able to view target computers on switch ports **1** and **3** while another user can view and interact with computers on switch ports **1** and **2**. In this manner, the system of the present invention can provide monitoring capabilities to less trusted individuals and full access to other, trusted individuals.

In an alternate embodiment, two or more different users may connect to the same controller **50**. In this embodiment, the two or more users may control different controller cards or may share access to the same controller card. In this embodiment, the captured GDI calls for a controller card are routed to the appropriate remote control software. Likewise, a user may be connected to multiple controller cards on one or more computers simultaneously. In that case, the user can monitor and control several target devices simultaneously.

Additional processing performed by the intelligent digitizer **75** is the analysis of the blocks. As shown in FIG. *7a*, the system maintains at least two status bits per block, although other status bits are also possible. The first status bit indicates which blocks have changed (either with or without filtering). This bit acts as a "dirty" bit in a cache. This bit can be separated into two bits if the system is to track which blocks have changed at all versus which blocks have changed more than the threshold. This threshold may be (1) global for the whole screen or (2) specific to particular

blocks. Moreover, this threshold may be updated dynamically either (1) at a user's request or (2) in response to an automatic adjustment of parameters to change performance characteristics.

The second bit illustrated indicates whether the corresponding block is a single color. As described above, if the block is a single color, then the block can be compressed by redrawing the block as a single GDI call, as discussed above.

As also discussed above, blocks can be compared for similarity to other blocks. Although not shown in the status fields of FIG. 7*a*, the status fields can include a reference to another block to which the current block is equal.

FIG. 7*b* shows a memory area that can be read by the microprocessor of the controller to determine which blocks have changed or are a single color. If additional bits of status information per block were used, the entry for each block would be widened by that number of bits.

In addition to indicating whether a block has changed, the intelligent digitizer **75** can also, in hardware, track which pixels within a block have changed. When tracking which pixels have changed, a memory area, as shown in FIG. **8**, is assigned to each block. The analyzing digitizer control application **240** can then read from memory the changed bits and determine if individual pixels should be redrawn of if the block should be redrawn in its entirety. By reading the first 32 bits of that memory and comparing with zero, the system can determine if any pixels in that line have changed. If not, the next line can be processed. In an alternate embodiment, the hardware contains a separate register for each block which identifies which lines within the block have changed. In this way, the system can quickly identify the lines that contain changed pixels.

Although the above description has focused on the normal operation of the present invention, the processing of the system may be paused when a user is temporarily uninterested in the changes on the target device. The analyzing digitizer control application **240** freezes its status in response to a message from the controlling computer. If the user has minimized the screen representing the target device on the monitor of the controlling computer **12**, then real-time updates of changes to the screen are not necessary. The internal buffers of the controller **50** that represent the last screen sent to the controlling computer **12** are no longer updated—i.e., they are frozen. However, the buffers representing the sampled video signals from the target device continue to be overwritten. The system then continues to track which blocks have changed in comparison to the frozen blocks—not in comparison to a previously sampled blocks. When the screen is re-enlarged, the controller **50** is unfrozen and the changes are sent back to the controlling computer **12**.

Thus, until the screen is un-minimized, the bandwidth that would have been used to send the changes (which would not have been seen) is saved. This is especially important when simultaneously monitoring multiple target devices over a lower-bandwidth modem connection. This method of performing comparison with the frozen blocks still allows the analyzing digitizer control application **240** to inform the controlling computer **12** of how many blocks have changed—without having to send those changes. Thus, the minimized icon on the controlling computer that represents the target device may flash or an audio signal may be played to inform the user that a major change to the screen has occurred.

In light of the inherent delay in the transmission process, the digitized mouse pointer on a target computer may be updated too slowly to allow accurate control of the mouse. As a result, the controlling computer **12** generates a pseudo-cursor (e.g., a set of cross-hairs) that indicates where the digitized cursor should be. To initialize this process, the digitizer control application **220** (or the analyzing digitizer control application **240**) sets the cursor of the target computer to a known location. For example, by sending to the target computer a series of mouse commands, it is possible to drive the cursor to the upper left hand-comer (the **0,0** comer), no matter where the cursor was prior the series of commands. The original cursor is then forced back down to be aligned with the cross-hairs.

As the mouse commands are received by the digitizer control application **220** (or the analyzing digitizer control application **240**), they are processed and passed on to the target device (which updates its local cursor). In order to avoid overloading the target computer with mouse packets, the digitizing control application **220** can queue mouse commands and end those mouse commands as a group. Alternatively, the digitizer control application **220** (or the analyzing digitizer control application **240**) can completely filter out a series of mouse movement events. To reproduce the effect that the filtered commands would have had, the system periodically samples the mouse position and sends, to the target controller, a mouse movement command representing the difference between the new position and the previous mouse position.

If the mouse pointer generated at the target controller **50** ever becomes out of alignment with the pointer generated on the target computer, the user can reset the pointers using a hot-key. Like during initialization, the target computer ig then sent a series of mouse commands to move the pointer to a known location and then from the known location to the position consistent with the cross-hairs drawn by the digitizer control application **220** (or the analyzing digitizer control application **240**). When the window of the digitizing control application **220** has the focus, this re-synchronization process is also performed when the mouse enters an active window of the digitizer control application **220** (or the analyzing digitizer control application **240**).

The above discussion has described the present invention in terms of remote control software **200** and an analyzing digitizer control application **240** that are separate software programs. In an alternate embodiment, the functionality of those two programs is more tightly integrated—either through the use of an API to communicate between them, or by combining the two into a single application. In this tighter integration, the analyzing digitizer control application **240** can transmit the changed blocks to the remote control software **200** in either compressed or uncompressed format. One example of a compressed format is a differential format in which a change flag indicates whether or not each pixel (or line) has changed. Then, the compressed block includes only the values within the block that have changed. Thus, the number of bytes to transmit is reduced as long as the overhead of the flags is less than the number of bytes saved by not transmitting those unchanged pixels in the block.

One implementation of such a compression header is shown in FIG. 9*a*. The header consists of 32 words that are each 32 bits—one bit for each pixel. As shown in the first line, three pixels in the first 32 pixels are changed. No other pixels in blocks **2–31** are changed, but in the last line, one additional pixel has changed. The data for the four pixels then follows the header.

A second implementation of the compression header utilizes a block header which indicates which lines have changed. The header indicates that only the first and last

lines have changed, so the bit flags for those lines are included—without including the bit flags for the unchanged lines.

Another compression technique used in an alternate embodiment includes encoding a block as (1) a reference to a known block (not necessarily the block from the previous screen capture) and (2) the changes that must be made to the referenced block in order to generate the current block. For example, if the background of an application changes, then all blocks identified as part of the background can be changed by simply referencing the first background block. If a portion of the block was not background, then only those parts that are not the background need to be encoded in the block. This technique similarly works for blocks that are almost completely one color. The block is simply encoded as (1) the background color of the block and (2) those pixels that are different from the background color.

In an alternate embodiment, in order to provide even further compression, blocks are compressed using intra-block compression. For example, a block may be compressed using run-length coding (with or without end-of-block markers) or Ziv-Lempel-Welch (LZW) encoding.

Although the target controller **50** has been described above as performing only the screen capture functions, that target controller **50** can provide additional functionality as well. The digitizer control application **220** and the analyzing digitizer control application **240** can be minimized so that the user can access the other programs stored on the target controller **50**. As such, the target controller **50** can be used to configure the network, cycle power to individual computers (**20***a* to **20***c*) and any other function that can be performed on computer to which a user is connected. It is even possible that the target controller **50** be connected to one of the switches that it samples.

In yet another embodiment of the present invention, the system captures outputs to a digital display rather than an analog display. In that embodiment, it is not necessary to convert from analog to digital format. The system simply buffers and analyzes the video data as if it were sampled data.

Obviously, numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that, within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. A target controller for remotely controlling at least one of an external target switch and an external target computer, the controller comprising:

an analog video interface for receiving analog video signals from the at least one of an external target switch and an external target computer;

a digitizer for digitizing the analog video signals received from the analog video interface;

a microprocessor;

a memory comprising plural computer code devices including:

a first computer code device configured to divide the digitized video signals into blocks of digitized video;

a second computer code device configured to compare a first block from a first frame to a second block from a subsequent frame; and

a third computer code device configured to send only pixel values that changed between the first and second blocks.

2. The controller as claimed in claim **1**, wherein the third computer code device comprises a fourth computer code device configured to detect if all pixels in a block are a single color such that the block can be redrawn using a single GDI call to fill a block.

3. The controller as claimed in claim **1**, wherein the second computer code device comprises a fourth computer code device configured to filter each block before determining if the block has changed.

4. The controller as claimed in claim **3**, wherein the fourth computer code device comprises a fifth computer code device configured to utilize a percentage threshold, wherein the block is not designated as changed if a number of changed pixels within the block is less than the percentage threshold.

5. The controller as claimed in claim **3**, wherein the fourth computer code device comprises a fifth computer code device configured to utilize an intensity threshold, wherein the block is not designated as changed if a pixel having a maximum change within the block has a change less than the intensity threshold.

6. The controller as claimed in claim **3**, wherein the fourth computer code device configured to filter each block comprises a fifth computer code device configured to dynamically change a filter used by the fourth computer code device.

7. The controller as claimed in claim **1**, wherein the analog video interface receives analog video signals from an external target switch, and wherein the first computer code device comprises a fourth computer code device configured to switch a current connection of the external target switch such that the analog video signals change from a first computer to a second computer.

8. The controller as claimed in claim **7**, further comprising:

a fifth computer code device configured to capture a mouse click and correlate the mouse click to one of plural windows; and

is a sixth computer code device configured to convert the mouse click into a series of switch commands that switch the external target switch to an external target computer corresponding to the one of plural windows.

9. The controller as claimed in claim **1**, further comprising:

one of a keyboard interface and a mouse interface; and

a fourth computer code device configured to send a command received from the one of a keyboard interface and a mouse interface to the at least one of an external target switch and an external target computer.

10. The controller as claimed in claim **1**, further comprising:

an integrated keyboard and mouse interface; and

a fourth computer code device configured to send a command received from the integrated keyboard and mouse interface to the at least one of an external target switch and an external target computer.

11. The controller as claimed in claim **1**, wherein the first computer code device comprises a fourth computer code device configured to dynamically change a size of the blocks of the digitized video.

12. The controller as claimed in claim **1**, wherein the third computer code device comprises a fourth computer code device configured to compress the changed pixel values.

13. The controller as claimed in claim **1**, wherein the fourth computer code device comprises a fifth computer code device configured to change a compression technique used by the fourth computer code device.

14. The controller as claimed in claim **1**, wherein the first computer code device comprises a fourth computer code

device configured to automatically determine a resolution of the analog video signals.

15. The controller as claimed in claim **14**, wherein the fourth computer code device further comprises a fifth computer code device configured to determine a delay to be used before sampling each line of video signals.

16. The controller as claimed in claim **1**, further comprising a fourth computer code device configured to detect a phase of the analog video signals based on signal jitter and to sample the analog video signals at substantially 180 degrees out of phase to the signal jitter.

17. The controller as claimed in claim **1**, further comprising a fourth computer code device configured to track mouse movements and output a pseudo-cursor independent of a digitized cursor.

18. The controller as claimed in claim **17**, further comprising a fifth computer code device configured to align the pseudo-cursor and the digitized cursor.

19. The target controller as claimed in claim **1**, wherein the external target switch comprises a KVM switch.

20. A target controller for remotely controlling at least one of an external target switch and an external target computer, the controller comprising:

an analog video interface for receiving analog video signals from the at least one of an external target switch and an external target computer;

a digitizer for digitizing the analog video signals received from the analog video interface;

a first logic device configured to divide the digitized video signals into blocks of digitized video;

a second logic device configured to compare a first block from a first frame to a second block from a subsequent frame; and

a third logic device configured to send only pixel values that changed between the first and second blocks.

21. The controller as claimed in claim **20**, wherein the first, second, and third logic devices are implemented as reconfigurable logic devices in a field programmable gate array.

22. The target controller as claimed in claim **20**, wherein the external target switch comprises a KVM switch.

23. A target controller for remotely controlling at least one of an external target switch and an external target computer, the controller comprising.

an analog video interface for receiving analog video signals from the at least one of an external target switch and an external target computer;

a digitizer for digitizing the analog video signals received from the analog video interface;

first means for dividing the digitized video signals into blocks of digitized video;

second means for comparing a first block from a first frame to a second block from a subsequent frame; and

third means for sending only pixel values that changed between the first and second blocks.

24. The target controller as claimed in claim **23**, wherein the external target switch comprises a KVM switch.

* * * * *

Appendix C

Evidence Appendix

Other than the reference attached to the Appeal Brief as Appendix B, no evidence was submitted pursuant to 37 C.F.R. §§ 1.130, 1.131, or 1.132, and no other evidence was entered by the Examiner and relied upon by Appellant in the Appeal.

Appendix D

Related Proceedings Appendix

1. Decision on Appeal (referenced hereinabove as the "*First Appeal Decision*") issued on May 24, 2005.

2. Decision on Request for Rehearing (referenced hereinabove as the "*Rehearing Decision*") issued on September 16, 2005.

Appendix D

Ref. 1.

The opinion in support of the decision being entered today was not written for publication and is not binding precedent of the Board.

DOCKETED Amendment/Rehearing Due: July 24, 2005

Paper No. 24

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Ex parte SHRINIWAS OHIA

Appeal No. 2005-0521
Application No. 09/436,920

ON BRIEF

MAILED MAY 24 2005 PAT. & T.M. OFFICE BOARD OF PATENT APPEALS AND INTERFERENCES

Before JERRY SMITH, LEVY and BLANKENSHIP, Administrative Patent Judges.

JERRY SMITH, Administrative Patent Judge.

DECISION ON APPEAL

This is a decision on the appeal under 35 U.S.C. § 134 from the examiner's rejection of claims 1-21, which constitute all the claims in the application.

The disclosed invention pertains to a method and apparatus for communicating information between a management card and first and second interface cards coupled to the management

Docket ✓ Wrapper ✓

RVF Docketed

Reference(s)

card. A client identifies an interface card or a network device associated with the interface card, and a communication link between the client and a particular one of the first and second interface cards is established in response to the interface card identified by the client.

Representative claim 1 is reproduced as follows:

> 1. A system for communicating management information, comprising:
>
> a first interface card;
>
> a second interface card; and
>
> a management card coupled to the first interface card and the second interface card, the management card operable to:
>
> receive a command from a client, the command identifying an interface card or a network device associated with an interface card;
>
> establish a communication link between the client and a particular one of the first interface card and the second interface card selected in response to the command communicated by the client; and
>
> communicate management information using the communication link.

The examiner relies on the following references:

Flood et al. (Flood)	4,937,777	Jun. 26, 1990
Schneider et al. (Schneider)	6,304,895	Oct. 16, 2001 (filed Jul. 23, 1999)

Claims 1, 4-7, 10-14, 16 and 18-21 stand rejected under 35 U.S.C. § 102(e) as being anticipated by the disclosure of Flood. Claims 2, 3, 8, 9, 15 and 17 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over the teachings of Flood in view of Schneider.

Rather than repeat the arguments of appellant or the examiner, we make reference to the briefs and the answer for the respective details thereof.

OPINION

We have carefully considered the subject matter on appeal, the rejections advanced by the examiner and the evidence of anticipation and obviousness relied upon by the examiner as support for the rejections. We have, likewise, reviewed and taken into consideration, in reaching our decision, the appellant's arguments set forth in the briefs along with the examiner's rationale in support of the rejections and arguments in rebuttal set forth in the examiner's answer.

It is our view, after consideration of the record before us, that the evidence relied upon supports the examiner's rejection of claims 1, 2, 4-8, 10-16 and 18-21, but the evidence does not support the rejection of claims 3, 9 and 17. Accordingly, we affirm-in-part.

We consider first the rejection of claims 1, 4-7, 10-14, 16 and 18-21 under 35 U.S.C. § 102(e) as being anticipated by Flood. Appellant has indicated that claims 1, 4, 5, 7, 10-12, 14, 16, 18, 19 and 21 stand or fall together as a first group, and claims 6, 13 and 20 stand or fall together as a second group (brief, page 5). Consistent with this indication appellant has made no separate arguments with respect to any of the claims on appeal within each group. Accordingly, all the claims within each group of claims will stand or fall together. Note In re King, 801 F.2d 1324, 1325, 231 USPQ 136, 137 (Fed. Cir. 1986); In re Sernaker, 702 F.2d 989, 991, 217 USPQ 1, 3 (Fed. Cir. 1983). Therefore, we will consider the anticipation rejection against claims 1 and 6 as representative of all the claims rejected on anticipation.

With respect to independent claim 1, the examiner has indicated how he finds the claimed invention to be fully met by the disclosure of Flood (answer, page 4). Appellant argues that given the elements identified by the examiner, Flood fails to disclose a management card operable to receive a command from a client as claimed. Appellant also argues that nowhere does Flood state that its command identifies an interface card or a network device associated with an interface card as claimed. Appellant

additionally argues that Flood fails to disclose that the management card is operable to establish a communication link between the client and a particular one of the interface cards in response to the command from the client. Appellant asserts that connections provided by cables 25 and 26 in Flood are not in response to a command communicated by the client (brief, pages 7-10).

The examiner responds that Flood teaches that a central host computer can program and control the operation of a plurality of programmable controllers. The examiner notes that the claimed commands from a client are met by the programming instructions from a user in Flood. The examiner also notes that system controller 16 in Flood is connected to the programming terminal to receive user programs (answer, pages 9-11).

Appellant responds that the programming instructions of Flood have nothing to do with identifying an interface card or a network device associated with an interface card as claimed. Appellant argues that, instead, the programming instructions of Flood are used for programming and controlling a plurality of programmable controllers. Appellant argues that the examiner is relying on two different portions of Flood to teach the claimed

command. Appellant repeats his position that system controller 16 of Flood cannot be the management card and meet the recitations of claim 1 (reply brief, pages 2-5).

We will sustain the examiner's rejection of representative claim 1 and of the other claims grouped therewith. Although the examiner has not properly identified what elements of Flood correspond to the claimed first interface card, second interface card and management card, it appears from the record that the first and second interface cards can be any of the processors 18 or scanners 20, and the management card corresponds to the system controller 16. It is clear that system controller 16 of Flood is coupled to each of the processors 18 and scanners 20. Claim 1 recites that the management card (system controller 16) receives a command from a client identifying an interface card or network device associated with an interface card. Flood meets this recitation because system controller 16 receives commands from a client (24) which identify the programable controller to which they are intended. The management card (system controller 16) in Flood establishes a communication link to the designated controller so that the programs and commands from terminal 24 can be downloaded to the appropriate controller. Finally, Flood meets the final step of claim 1 because the system

controller 16 uses the established link to communicate management information between the terminal 24 and the designated programmable controller. Note that each of the execution processors of Flood is individually loaded with user control programs (column 5, lines 1-5).

Although we have given careful consideration to appellant's arguments in the briefs, it appears to us that appellant has not fully grasped the manner in which the examiner has read claim 1 on the disclosure of Flood. Although appellant argues that system controller 16 receives no command, it is clear to us that system controller 16 receives a command from terminal 24 indicating which programmable controller is to be accessed by the system controller. Although cable 25 is not connected in response to a command, the appropriate connection along the control, data and address lines 21-23 between the system controller 16 and the designated programmable controller is made in response to the commands received on cable 25 from the client. We find that this operation meets the recitations of claim 1. In other words, individual communication links are established between system controller 16 and execution processors 18 on lines 21-23 in response to requests made by the client at terminal 24.

With respect to representative claim 6, the examiner has indicated how he reads the claimed invention on the disclosure of Flood (answer, page 5). We note that appellant grouped claims 6, 13 and 20, which are rejected under 35 U.S.C. § 102, with claims 3, 9 and 17, which were rejected under 35 U.S.C. § 103. Appellant has only addressed the limitations of claim 3 and has not made arguments with respect to the limitations of claim 6 even though claim 6 is broader than claim 3. Since claims 6, 13 and 20 each recites a limitation which has not been argued by appellant, these claims stand or fall with the claim from which they each respectively depend. Since we have sustained the rejection with respect to each of the parent claims, we also sustain the rejection of claims 6, 13 and 20.

We now consider the rejection of claims 2, 3, 8, 9, 15 and 17 under 35 U.S.C. § 103 based on the teachings of Flood and Schneider. In rejecting claims under 35 U.S.C. § 103, it is incumbent upon the examiner to establish a factual basis to support the legal conclusion of obviousness. See In re Fine, 837 F.2d 1071, 1074, 5 USPQ2d 1596, 1598 (Fed. Cir. 1988). In so doing, the examiner is expected to make the factual determinations set forth in Graham v. John Deere Co., 383 U.S. 1, 17, 148 USPQ 459, 467 (1966), and to provide a reason why one

having ordinary skill in the pertinent art would have been led to modify the prior art or to combine prior art references to arrive at the claimed invention. Such reason must stem from some teaching, suggestion or implication in the prior art as a whole or knowledge generally available to one having ordinary skill in the art. Uniroyal, Inc. v. Rudkin-Wiley Corp., 837 F.2d 1044, 1051, 5 USPQ2d 1434, 1438 (Fed. Cir.), cert. denied, 488 U.S. 825 (1988); Ashland Oil, Inc. v. Delta Resins & Refractories, Inc., 776 F.2d 281, 293, 227 USPQ 657, 664 (Fed. Cir. 1985), cert. denied, 475 U.S. 1017 (1986); ACS Hosp. Sys., Inc. v. Montefiore Hosp., 732 F.2d 1572, 1577, 221 USPQ 929, 933 (Fed. Cir. 1984). These showings by the examiner are an essential part of complying with the burden of presenting a prima facie case of obviousness. Note In re Oetiker, 977 F.2d 1443, 1445, 24 USPQ2d 1443, 1444 (Fed. Cir. 1992). If that burden is met, the burden then shifts to the applicant to overcome the prima facie case with argument and/or evidence. Obviousness is then determined on the basis of the evidence as a whole and the relative persuasiveness of the arguments. See Id.; In re Hedges, 783 F.2d 1038, 1039-40, 228 USPQ 685, 686 (Fed. Cir. 1986); In re Piasecki, 745 F.2d 1468, 1472, 223 USPQ 785, 788 (Fed. Cir. 1984); and In re Rinehart, 531

F.2d 1048, 1052, 189 USPQ 143, 147 (CCPA 1976). Only those arguments actually made by appellant have been considered in this decision. Arguments which appellant could have made but chose not to make in the brief have not been considered and are deemed to be waived (see 37 CFR § 41.37(c)(1)(vii)(2004)).

Appellant has indicated that claims 2, 8 and 15 stand or fall together as a single group (brief, page 5). The examiner has indicated how he finds the invention of these claims to be unpatentable (answer, pages 6-8). Appellant argues that the examiner has failed to identify which components in Schneider correspond to the management card, the switch, the processor and the memory of claim 2. Appellant also argues that switches 74a and 74b do not form part of a management card, and the combination fails to teach the various elements as claimed (brief, pages 10-11).

The examiner responds that Schneider teaches how a plurality of users may control different cards and how different switch ports are selected (answer, pages 11-12). Appellant responds that switches 74a and 74b of Schneider do not form part of a management card and there is no teaching of a mapping to a first port and a second port (reply brief, page 5).

We will sustain the examiner's rejection of claims 2, 8 and 15. The examiner has cited Schneider to teach the claimed manner in which communication links are established between a client and a port. The examiner has explained in some detail how the configuration wizard in Schneider would have led the artisan to map specific communications in Flood to the appropriate port in the management card. Appellant has not addressed the specific explanation offered by the examiner but has simply asserted that elements 74a and 74b of Schneider do not meet the claimed invention. We can find nothing in the examiner's explanation which suggests that the examiner has relied on these elements to support his position.

Since the examiner appears to have established a prima facie case of the obviousness of claims 2, 8 and 15, and since appellant's arguments do not convince us that the rejection is in error, we sustain the rejection of these claims as noted above.

Although appellant nominally grouped claims 3, 9 and 17 with claims 6, 13 and 20, these claims are substantially different and were rejected on a different basis as discussed above. We will consider these claims separately because appellant argued them and because the examiner responded to the arguments with respect to these claims. The examiner's rejection

of these claims is set forth on page 9 of the answer. Appellant argues that the cited portions of Flood have nothing to do with the claimed processor which is operable to configure the management information for the operating system of the network device associated with the particular interface card. Appellant notes that Flood does not even remotely consider different operating systems (brief, pages 11-12). The examiner responds that the artisan would have understood that the first and second network devices of Flood use first and second operating systems (answer, page 13). Appellant responds that there is no basis to conclude that Flood teaches the limitations of claim 3. Appellant notes that the Flood-Schneider combination does not even discuss operating systems or different operating systems associated with different network devices (reply brief, page 5).

We will not sustain the examiner's rejection of claims 3, 9 and 17 for essentially the reasons argued by appellant in the briefs. Specifically, the applied prior art makes no mention of first and second operating systems and of configuring management information for the operating system. The examiner's "finding" of different operating systems in Flood is nothing more than speculation and has no support in the reference. While we suspect that different devices can operate under different

operating systems, and that communicated information must be configured based on the operating system, we are not permitted to substitute our opinions or beliefs for evidence lacking in the record. The examiner is required to provide a clear evidentiary record to support the rejection of each claim. The prior art applied in the examiner's rejection simply does not provide the support needed to reject claims 3, 9 and 17.

In summary, the examiner's anticipation rejection has been sustained with respect to all claims, and the obviousness rejection has been sustained with respect to claims 2, 8 and 15, but has not been sustained with respect to claims 3, 9 and 17. Therefore, the decision of the examiner rejecting claims 1-21 is affirmed-in-part.

No time period for taking any subsequent action in connection with this appeal may be extended under 37 CFR § 1.136(a)(1)(iv).

AFFIRMED-IN-PART

JERRY SMITH
Administrative Patent Judge

STUART S. LEVY
Administrative Patent Judge

HOWARD B. BLANKENSHIP
Administrative Patent Judge

) BOARD OF PATENT APPEALS AND INTERFERENCES

JS:hh

BAKER & BOTTS, LLP
2001 ROSS AVE.
DALLAS, TX 75201-2980

Appendix D

Ref. 2.

062891.0320

The opinion in support of the decision being entered today was not written for publication and is not binding precedent of the Board.

Paper No. 26

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES

Ex parte SHRINIWAS OHIA

Appeal No. 2005-0521
Application No. 09/436,920

MAILED
SEP 16 2005
U.S. PATENT AND TRADEMARK OFFICE
BOARD OF PATENT APPEALS
AND INTERFERENCES

ON BRIEF

Before JERRY SMITH, LEVY, and BLANKENSHIP, Administrative Patent Judges.

JERRY SMITH, Administrative Patent Judge.

ON REQUEST FOR REHEARING

Appellant requests that we reconsider that portion of our decision of May 24, 2005 wherein we sustained the rejection of claims 1, 4-7, 10-14, 16 and 18-21 as unpatentable under 35 U.S.C. § 102(e) and the rejection of claims 2, 8 and 15 as unpatentable under 35 U.S.C. § 103(a).

Docket ✓ Wrapper ✓
RVF Docketed
Reference(s)

Appellant asserts that we misinterpreted the teachings of the Flood reference in our original decision which is grounds for reversing our decision with respect to the claims noted above.

We have reconsidered our decision of May 24, 2005 in light of appellant's comments in the request for rehearing, and we find no error therein. We, therefore, decline to make any changes in our prior decision for the reasons which follow.

Appellant argues that there is no "communication link" between terminal 24 and processing modules 18. Appellant bases this argument on his position that terminal 24 and program execution modules 18 cannot concurrently access buses 31-33 of system controller 16. Thus, appellant argues that Flood never establishes a communication link between terminal 24 and processors 18. In other words, appellant's argument is apparently based on appellant's view that a "communication link" requires that the complete path from terminal 24 to processors 18 must exist at the same time.

We note that appellant's arguments with respect to this feature of the claimed invention in the briefs were based on whether there was a connection between terminal 24 and processors 18. We essentially found that terminal 24 was connected to processors 18. We now find that the fact that data in Flood is

sent from terminal 24 to processors 18 is sufficient to meet the broadest reasonable interpretation of the term "communication link." Appellant had the opportunity in the briefs to argue the meaning of the term "communication link," but failed to do so. Whether the examiner and the Board properly interpreted the term "communication link" is a question of fact which should have been argued before the examiner and should be part of the record we are now reviewing. For purposes of this decision, and based on the record of prosecution, we simply find that a communication link exists between the terminal 24 and the processors 18 of Flood because information is transferred between them.

We have carefully considered the arguments raised by appellant in the request for rehearing, but we can find no errors in our original decision. We are still of the view that the invention set forth in claims 1, 2, 4-8, 10-16 and 18-21 is unpatentable over the applied prior art.

We have granted appellant's request to the extent that we have reconsidered our decision of May 24, 2005, but we deny the request with respect to making any changes therein.

No time period for taking any subsequent action in connection with this appeal may be extended under 37 CFR § 1.136(a).

REHEARING - DENIED

JERRY SMITH
Administrative Patent Judge

STUART S. LEVY
Administrative Patent Judge

HOWARD B. BLANKENSHIP
Administrative Patent Judge

) BOARD OF PATENT APPEALS AND INTERFERENCES

JS:hh

Appeal No. 2005-0521
Application No. 09/436,920

BAKER & BOTTS, LLP
2001 ROSS AVE.
DALLAS, TX 75201-2980